POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

Progetto e sviluppo di un sistema di input manipolabile per curve 3D

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Vincenzo CAGLIOTI
Correlatore: Ing. Alessandro GIUSTI

Tesi di Laurea di:
Lorenzo MUREDDU, matricola 669104

Anno Accademico 2006-2007

A Ornella.

Sommario

La corretta gestione dell'*interazione Uomo-Macchina* rappresenta uno dei principali problemi dell'informatica. Suddetta gestione può avvenire realizzando interfacce semplici e *User-Friendly* che permettano di semplificare l'utilizzo delle complicate funzionalità dei moderni computer.
Nell'area della computer graphics è particolarmente sentita la necessità di sviluppare strumenti per modellare complesse strutture tridimensionali, in quanto strumenti "classici "come mouse e tastiera non sono sufficenti a gestire gli oggetti 3d in maniera ottimale.
Lo scopo del presente lavoro di tesi è quello di fornire uno strumento semplice ed immediato per la creazione di curve 3d semplici o complesse. É stato, infatti, realizzato un programma in grado di ricostruire curve in un ambiente di modellazione 3d, partendo dall'immagine di un tubo fotografata in remoto.
Per ottenere tale risultato sono stati utilizzati diversi strumenti, quali Matlab, linguaggio di programmazione matematico, e Blender, software di modellazione 3d. Questi due software sono stati interfacciati in modo da comunicare reciprocamente. Nella realizzazione del progetto di tesi si è usufruito di un algoritmo di ricostruzione 3d pre-esistente scritto in Matlab. Grazie al sistema di librerie Python che Blender offre, suddetto algoritmo è stato integrato in modo da poter sfruttare la curva ricostruita all'interno del software di modellazione 3d.
L'applicativo prodotto fornisce un valido strumento che permette all'utente di ottenere una curva, complessa a piacere, in un software di modellazione 3d, semplicemente modellando fisicamente un tubo.

Ringraziamenti

Ringrazio Alessandro Giusti per avermi seguito egregiamente durante tutto l'arco del lavoro di tesi, mostrando entusiasmo per le mie conquiste.
Ringrazio tutti i compagni di corso con cui ho condiviso questi tre anni assieme e soprattutto Andrea, Luca e Mirons che considero dei veri amici, capaci di sopportare i miei discorsi deliranti e i miei sbalzi d'umore.
Ringrazio le mie meravigliose zie, Anna e Pietrina per il supporto nei momenti difficili e sì, ringrazio anche mio Padre, perchè, nonostante tutte le difficoltà, senza di lui non sarei quì, vi voglio bene.
Un sentito grazie va anche a Giancarlo per il supporto logistico che in svariate occasioni mi ha salvato il fondoschiena.
Ma soprattutto ringrazio il mio amore, Ornella, senza di lei non so quanto sarei andato avanti in questi anni. Con la sua forza mi ha spronato innumerevoli volte e mi ha fatto conoscere la vera vita. Sono felice di svegliarmi ogni mattina con lei, di avere un rapporo che è solo nostro, fatto della condivisione totale di ogni cosa. Grazie Amore.

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'interazione Uomo-Macchina rappresenta un'aspetto fondamentale dello sviluppo dell'informatica. Negli ultimi decenni i computer sono entrati in ogni casa e nella vita di ognuno di noi; ciò ha portato l'informatica a trasformarsi da strumento complicato per pochi utenti esperti a tecnologia alla portata di tutti. Questo processo di cambiamento ha esortato nel tempo allo studio e allo sviluppo di numerosi sistemi di input basati sulla semplicità di apprendimento e di usabilità da parte dell'utente. Si parla recentemente di applicazioni e periferiche di input *User-Friendly* proprio per identificare questi concetti di semplicità nell'utilizzo del software.

La grafica tridimensionale, grazie ad un esponenziale aumento delle prestazioni dei computer, si è radicalmentem evoluta in questi anni, tanto da giocare un ruolo sempre più di maggior rilievo nell'utilizzo dei computer, grazie all'industria dei videogiochi e del cinema. In luce di questo sviluppo, il mercato ha cominciato a richiedere nuovi e migliori strumenti per sfruttare appieno le potanzialità dei moderni motori grafici 3d. I soli mouse e tastiera, infatti, risultano essere strumenti poco consoni, ma soprattutto molto scomodi, per la creazione e la manipolazione di complesse strutture 3d.

Lo scopo della tesi è proprio quello di fornire un efficace strumento per creare e manipolare entità tridimensionali, nel caso specifico curve spaziali, garantendo la facilità di apprendimento e di utilizzo anche per un utente non esperto. Più nello specifico, si è pensato di fornire un'applicazione che permettesse all'utente di creare curve 3d in un software di modellazione 3d, manipolando fisicamente un tubicino che potesse mantenere la forma desiderata.

Nella pratica è stato sviluppato un sistema che:

1. fotografa in remoto un tubicino con anima in fil di ferro, attraverso

una fotocamera digitale connessa al computer;

2. ricostruisce la curva 3d dalla singola immagine scattata, utilizzando l'algoritmo di ricostruzione delle *canal surface* in [25];

3. importa e disegna in Blender, un famoso software di modellazione 3d open-source, la curva 3d ricostruita.

Il tutto viene governato da una semplice interfaccia grafica disegnata all'interno di Blender stesso.

Blender, in quanto programma di modellazione 3d, permette di creare e gestire complesse strutture tridimensionali, realizzando rendering di animazioni e immagini, o applicazioni ludiche in 3d.

Il presente lavoro è stato realizzato a partire dall'algoritmo in [25] scritto in Matlab [22], linguaggio matematico di programmazione. Suddetto algoritmo è stato in principio integrato con Blender [21] per la visualizzazione e la manipolazione della curva 3d, e successivamente con la libreria per Matlab *Camera Box* [24], per la gestione della fotocamera da remoto.

Nell'applicativo sviluppato è stata inserita un'interessante funzionalità: la possibilità di innestare curve su altre curve secondo tre parametri definiti dagli angoli α, β e γ. Ciò permette di creare strutture multi-curve complesse a piacere.

L'integrazione con Blender permette di sfruttare al meglio le curve 3d ricostruite. All'interno del software di modellazione 3d è infatti possibile usare la curva in diversi modi, tra cui:

- come oggetto stesso in un rendering, impostando alla curva una superfice che la avvolga;
- come percorso per l'animazione di un oggetto;
- come guida per deformare un oggetto.

Il lavoro svolto soddisfa appieno gli obbiettivi iniziali, mantenendo aperti possibili sviluppi futuri. Si potrebbe, ad esempio, ricostruire superfici attraverso strutture di tubicini a griglia, implementare applicazioni ludiche, quali ad esempio la creazione di montagne russe virtuali con la forma della curva ricostruita, oppure sfruttare il Game Engine di Blender e realizzare viodegiochi stand-alone.

Il software realizzato in questa tesi appartiene alla classe delle cosìdette *Tangible Interface*, termine che identifica strumenti di input non convenzionali per la soluzione di problemi relativi all'interfacciamento Uomo-Macchina.

Esempi di Tangible Interface sono [5], [8], [10], [12] e [13].

Quando applicate nell'ambito della grafica 3d le interfacce tangibili identificano, spesso, il processo di modellazione di particolari strutture, mediante l'utilizzo di strumenti materiali, come ad esempio l'utilizzo di mattoncini simil Lego o plastilina in [13].

Per quanto rigurda la ricostruzione 3d, l'algoritmo usato [25] è un esempio di *shape from contour*, [17] e [18], in quanto il primo passo della ricostruzione opera attraverso l'estrazione dei contorni nell'immagine. Questo processo è reso possibile in primis dall'effetto prospettico intrinseco nella foto, ma soprattutto da una precisa conoscenza del modello delle *canal-surface*, superfici a forma di tubo che identificano una curva.

Aspetto estremamente rilevante della tesi è indubbiamente lo studio e la manipolazione delle espressioni matematiche che caratterizzano le curve, quali *spline*[1], *curve Bézier*[2] e *NURBS*[3]. Nello specifico è stato implementato un metodo matematico per la trasformazione del formato spline, utilizzato all'interno di Matlab, nel formato di curva Bézier, interno a Blender. Ciò è stato possibile conoscendo la formulazione in forma polinomiale sia della spline che della curva Bézier.

La libreria Matlab CameraBox[24] introdotta sopra, sfruttando le librerie della Canon chiamate Canon SDK [20], rappresenta lo strumento usato per comunicare con i driver della fotocamera, in modo da poter scattare la fotografia in remoto.

Problema fondamentale nello sviluppo del progetto è stato interfacciare Matlab e Blender.
Blender è estendibile attraverso script Python [29] e, a questo scopo, fornisce una libreria [30] che permette di manipolare ogni oggetto creabile in Blender. Questo fatto ha permesso di trasformare il problema di coordinamento dei due software in un problema di coordinamento di due diversi linguaggi di programmazione, Matlab e Python. Il problema è stato risolto con l'ausilio di una libreria *wrapper* tra i due linguaggi chiamata MatlabWrap [27]. Per libreria wrapper si intende una libreria che permetta di richiamare funzioni scritte in un linguaggio, in questo caso Matlab, in un programma scritto in un altro linguaggio, Python.

Nella gestione della curva 3d sono state largamente adoperate le matrici di rototraslazione. Si è, infatti, realizzato un sistema che permettesse all'utente di avere il pieno controllo della curva selezionando sia un punto nello spazio che tre angoli che ne definiscono la rotazione iniziale.

Riassumendo, nell'ambito di questo lavoro di tesi è stato progettato e sviluppato un sistema per creare e manipolare curve 3d all'interno di un software di modellazione 3d, chiamato Blender. Questo è stato fatto utilizzando

vari strumenti:

- Camera Box [24], che ha permesso di scrivere una funzione Matlab per la gestione dello scatto da remoto di una fotocamera digitale;
- algoritmo in Matlab di ricostruzione 3d [25] a partire da singola immagine;
- libreria Python MatlabWrap [27] che ha permesso di coordinare Matlab e Blender;
- API di Blender per Pyhton, che ha permesso di programmare oggetti 3d all'interno del software stesso.

Come già detto gli utilizzi del software sono molteplici, dalla creazione di animazioni 3d, alla creazione di videogiochi e comunque tutti gli utilizzi, compreso il disegno industriale, per cui sia utile ottenere una curva 3D.

La tesi è strutturata nel modo seguente.

Nel capitolo due, dal titolo "Stato dell'arte", si mostra il contesto in cui il lavoro di tesi è inserito. Nello specifico, viene introdotta la teoria alla base della formulazione matematica delle curve 3d, descrivendo i formati spline, curve bézier e NURBS. Successivamente si mostrano esempi di interfacce tangibili, introducendo gli strumenti di input usati nella navigazione e nella modellazione 3d, quali periferiche di input a 6 gradi di libertà.

Nel capitolo tre, dal titolo "Impostazione del problema di ricerca", si descrivono gli strumenti sia software che fisici utilizzati nel progetto, argomentando inoltre le motivazioni che hanno portato alla scelta di tali strumenti.

Nel capitolo quattro, dal titolo "Progetto logico della soluzione del problema", si illustrano i passaggi logico-matematici sviluppati per risolvere il problema posto. Particolare attenzione viene prestata agli aspetti matematici nel controllo delle caratteristiche della curva, quali rotazione iniziale e formato.

Nel capitolo cinque, dal titolo "Aspetti Implementativi", vengono descritti approfonditamente gli aspetti implementativi, appunto, del progetto realizzato. Si analizzano per esteso, quindi, tutte le librerie utilizzate.

Nel capitolo sei, dal titolo "Realizzazioni sperimentali e valutazione", vengono visualizzate, anche con l'ausilio di immagini, le prove sperimentali svolte ed i possibili utilizzi del software all'interno di Blender.

Nelle conclusioni si riassume il lavoro svolto e si ipotizzano ulteriori futuri sviluppi del progetto.

Nell'appendice A si riporta tutto il listato, composto da due funzioni Matlab e da quattro script Python.

Nell'appendice B si mostra un abbozzo di manuale utente contenente le istruzioni per installare l'applicativo.

Nell'appendice C, infine, si descrivono alcuni esempi di utilizzo del software prodotto, mostrando alcuni rendering di immagini ed animazioni.

Capitolo 2

Stato dell'arte

All'interno del secondo capitolo si introducono gli argomenti affrontati e si illustrano progetti simili a quello qui sviluppato. Si predispongono, quindi, le basi teoriche per la comprensione degli argomenti trattati.

Il capitolo, suddiviso in tre parti, illustra:

- primariamente la teoria che sta alla base della rappresentazione matematica delle curve, sia in formato spline che in formato Bézier;
- in secondo luogo vengono illustrati alcuni metodi di input per ambienti grafici 3d, da input basati sulla realtà virtuale ai cosìdetti "Tangible" input basati su strumenti esterni per la modellazione 3d;
- infine viene illustata brevemente la teoria sottostante la ricostruzione 3d di oggetti o ambienti, comunemente nota con il termine "Shape from X", dove X rappresenta una specifica tecnica di ricostruzione, ad esempio *contuor*, *motion*, *shading*...

2.1 Rappresentazione matematica delle curve

In questa prima sezione si introduce l'oggetto matematico di funzione *Spline*, come rappresentazione matematica di una linea nello spazio tridimensionale, si discute della sua applicazione nella grafica, chiamata curva Bézier, e si illustra brevemente la generalizzazione di quest'ultima con le NURBS (Non-Uniform Rational B-Splines).

2.1.1 Spline

In analisi matematica, una spline è una funzione costituita da un insieme di polinomi raccordati tra loro, il cui scopo è interpolare in un intervallo un insieme di punti in modo da essere continua in ogni punto dell'intervallo. Suddetti punti sono chiamati nodi della spline. La funzione spline deve essere continua in ogni suo punto fino ad un ordine di derivata predefinito.

Data la suddivisione dell'intervallo chiuso $[a, b]$:

$$\Lambda \equiv \{a = x_0 < x_1 < \cdots < x_n = b\} \tag{2.1}$$

Si definisce funzione *spline* di grado p, con nodi nei punti x_i, con $0 \leq i \leq n$ dove $i \in \mathbb{N}$, una funzione su $[a, b]$ indicata con $s_p(x)$ tale che, nell'intervallo $[a, b]$ si abbia:

1. in ogni sottointervallo $[x_i, x_{i+1}]$ con $0 \leq i \leq n-1$, dove $i \in \mathbb{N}$,la funzione $sp(x)$ è un polinomio di grado p.
2. la funzione $s_p(x)$ e le sue prime $p-1$ derivate sono continue.

Una funzione spline viene definita *interpolante* se passa per ognuno dei punti che la definiscono. In particolare, dato il campionamento di una funzione $y = f(x)$ nei punti (xi, yi), con $0 \leq i \leq n$ e $i \in \mathbb{N}$, è detta spline interpolante la spline $s_p(x)$ tale che:

$$s_p(x_i) = y_i \ \forall i \tag{2.2}$$

La funzione spline di grado p $s_{p,j}(x)$ definita sul sottointervallo $[x_j, x_{j+1}]$, dove $0 \leq j \leq n-1$ e $j \in \mathbb{N}$, può sempre essere espressa nella forma:

$$s_{p,j}(x) = \sum_{i=0}^{p} a_{ij}(x - x_j)^i \tag{2.3}$$

dove gli $n(p+1)$ coefficienti a_{ij} sono da determinare imponendo le condizioni di continuità di $s_p^{(k)}$, dove k è l'ordine di derivata nei nodi interni:

$$s_{p,j-1}^{(k)}(x_j) = s_{p,j}^{(k)}(x_j) \quad \forall j, k \tag{2.4}$$

dove $0 \leq j \leq n-1$, $0 \leq k \leq p-1$ e $j, k \in \mathbb{N}$. Ciò però da luogo solo a $p(n-1)$ equazioni, pertanto il sistema di equazioni così ottenuto ha $n+p$ gradi di libertà.

Anche nel caso delle spline interpolanti, imponendo il passaggio della spline per i punti $(x_i, y_i) \quad i = 0 \ldots n$, non si è ancora in grado di determinare $p-1$ coefficienti. Per questo motivo nella pratica si è soliti aggiungere delle condizioni aggiuntive, in maniera che il sistema abbia soluzione unica.

Le condizioni aggiuntive più usate sono di due tipi:

- $s_p^{(k)}(a) = s_p^{(k)}(b)$ per $k = 1, \ldots, p-1$, dove a e b sono rispettivamente il primo e l'ultimo punto della spline. Questa condizione impone, cioè, che la spline abbia i valori delle derivate iniziali uguali a quelli delle derivate finali, di qualsiasi ordine esse siano. Le splines che soddisfano questo tipo di condizioni sono dette *spline periodiche*.

- $s_p^{(m+j)}(a) = s_p^{(m+j)}(b) = 0$ con $j = 0, \ldots, m-2$, e dove sia $p = 2m-1$ con $m \geq 2$. Questa condizione impone, cioè, che la spline inizi e finisca con valori di derivate di ordine maggiore di m uguali a 0. Le splines che soddisfano condizioni di questo tipo sono dette *spline naturali*.

Si ottengono così tutte le condizioni che permettono di definire una spline interpolante. Per maggiori informazioni si veda [1].

Nella pratica le funzioni spline vengono usate per interpolare punti nello spazio a 2 o a 3 dimensioni, o per definire delle curve ex novo. Per ottenere una spline tridimensionale bisogna considerare una equazione del tipo 2.3 per ogni coordinata spaziale, X,Y e Z, dove l'ascissa t viene definita dall'ascissa curvilinea:

$$s_{p,j,k}(t) = \sum_{i=0}^{p} a_{ij}(t - t_j)^i \qquad \forall k = X, Y, Z \tag{2.5}$$

2.1.2 Curva Bézier

Per disegnare curve in programmi di grafica a 2 o 3 dimensioni viene utilizzata una forma più comoda dello stesso oggetto matematico definito dalla spline, la *curva Bézier*.

Le curve Bézier devono il loro nome all'ingegnere francese della Renault Pierre Bézier che nel 1962 le usò per disegnare le carrozzerie delle automobili. Esse vengono definite da soli punti nello spazio 3d o 2d.

Esistono diverse categorie di curve Bézier, dipendemente dal grado dei polinomi che la definiscono:

Curve Bézier Lineari : Dati i punti P_0 e P_1, una curva Bézier lineare è una linea retta che li attraversa. La curva è data da:

$$\mathbf{B}(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \ , \ t \in [0,1] \tag{2.6}$$

Una curva Bézier lineare è visualizzabile in figura 2.1.

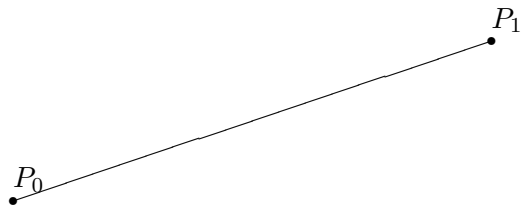


Figura 2.1: Curva Bézier Lineare

Curve Bézier Quadratiche : Una curva Bézier quadratica è il percorso tracciato tramite la funzione $B(t)$, dati i punti P_0, P_1, e P_2,

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2 \ , \ t \in [0,1]. \tag{2.7}$$

In figura 2.2 è rappresentato un esempio di curva Bézier quadratica.

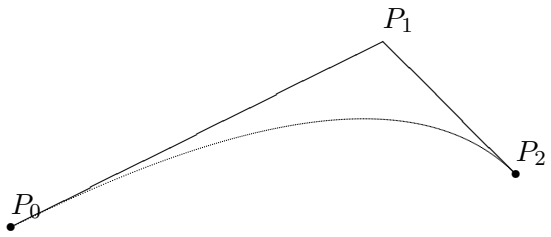


Figura 2.2: Curva Bézier Quadratica

Curve Bézier Cubiche : Una curva Bézier cubica è definita da quattro punti, P_0, P_1, P_2 e P_3, nel piano o in uno spazio tridimensionale. La curva ha inizio in P_0 si dirige verso P_1 e finisce in P_3 arrivando dalla direzione di P_2. In generale, essa non passa dai punti P_1 o P_2; questi punti sono necessari solo per dare alla curva informazioni direzionali. La distanza tra P_0 e P_1 determina quanto la curva si muove nella

direzione di P_2 prima di dirigersi verso P_3. Esempio di curva Bézier cubica in figura 2.3.
La forma parametrica della curva è:

$$\mathbf{B}(t) = \mathbf{P}_0(1-t)^3+3\mathbf{P}_1t(1-t)^2+3\mathbf{P}_2t^2(1-t)+\mathbf{P}_3t^3 \ , \ t \in [0,1]. \quad (2.8)$$

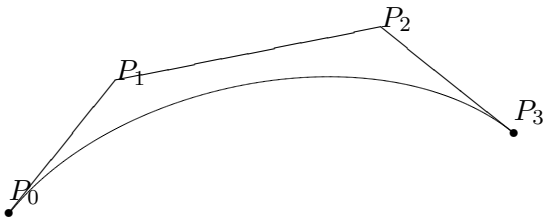


Figura 2.3: Curva Bézier Cubica

La curva Bézier di grado n può essere generalizzata come segue. Dati i punti P_0, P_1,..., P_n, la curva Bézier è:

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} \mathbf{P_i}(1-t)^{n-i}t^i \ , \ t \in [0,1]. \quad (2.9)$$

Per esempio, per n = 4:

$$\mathbf{B}(t) = \mathbf{P}_0(1-t)^4+4\mathbf{P}_1t(1-t)^3+6\mathbf{P}_2t^2(1-t)^2+4\mathbf{P}_3t^3(1-t)^1+1\mathbf{P}_4t^4 \ , \ t \in [0,1]. \quad (2.10)$$

I polinomi nell'equazione 2.9 di tipo:

$$\binom{n}{i}(1-t)^{n-i}t^i \quad (2.11)$$

sono conosciuti come *polinomi di base di Bernstein* di grado n e sono definiti da:

$$b_{i,n}(t) := \binom{n}{i}t^i(1-t)^{n-i}, \qquad i = 0,\dots n. \quad (2.12)$$

definito $0^0 = 1$.
Dove $\binom{n}{i}$ è chiamato *coefficente binomiale* e vale:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \quad (2.13)$$

I punti P_i sono chiamati *punti di controllo* per la curva Bézier. Il poligono formato connettendo i punti attraverso linee rette, iniziando da P_0 e finendo con P_n è chiamato *poligono Bézier*, ed esso contiene la curva Bézier.

In generale le curve Bézier possiedono alcune proprietà:

1. La curva inizia in P_0 e termina in P_n; questa è chiamata la proprietà della *interpolazione di punto finale*.

2. La curva è una linea retta se e solo se tutti i punti di controllo giacciono sulla curva, similmente, la curva Bézier è una linea retta se e solo se i punti di controllo sono *collineari*.

3. L'inizio della curva è tangente al primo lato del poligono Bézier, similmente la fine della curva è tangente all'ultimo lato del poligono Bézier.

4. Una curva può essere spezzata in qualsiasi punto in 2 sottocurve, o in un arbitrario numero di sottocurve, ognuna delle quali è essa stessa una curva Bézier.

5. Un cerchio non può essere esattamente formato da una curva Bézier, come neanche un arco di cerchio. Comunque una curva Bézier è un'adeguata approssimazione di un arco circolare abbastanza piccolo.

L'algoritmo fondamentale per il calcolo delle curve Bézier è chiamato *algoritmo di de Casteljau*. Esso è stato sviluppato nel 1959 da Paul de Casteljau, ingegnere della Citroën. L'algoritmo si basa sulla quarta proprietà precedentemente elencata ed è un metodo di calcolo ricorsivo. L'idea centrale del metodo è che suddividendo consecutivamente una curva Bézier i punti di controllo delle curve risultanti tendono alla curva stessa, in altre parole suddividendo più volte una curva Bézier si ottiene un'approssimazione della curva stessa definita dai punti di controllo.

Le curve Bézier sono largamente usate nella computer grafica per modellare curve smussate. Dato che la curva è contenuta completamente nell'insieme convesso dei suoi punti di controllo, i punti possono essere visualizzati graficamente ed usati per manipolare la curva intuitivamente. Trasformazioni geometriche come traslazione e rotazione possono essere applicate alla curva applicando le rispettive trasformazioni sui punti di controllo della curva.

Le più importanti curve Bézier sono le quadratiche e cubiche. Curve di grado più alto sono molto più costose da valutare. Quando sia necessario realizzare forme più complesse, più curve di secondo o terzo ordine sono

"incollate"insieme in forma di *spline Bézier*.

Per ulteriori informazioni sulle curve Bézier si veda [2].

2.1.3 NURBS

Le curve NURBS (Non-Uniform Rational B-Splines) rappresentano una generalizzazione delle curve Bézier che possono essere viste come *uniform non-rational B-splines*. Tutte le informazioni relative possono essere trovati in [3].

Una curva NURBS è definita dal suo *ordine*, da un'insieme di *punti di controllo* pesati, e da un *vettore di nodi*.

ordine : l'ordine di una curva NURBS definisce quanti punti di controllo nelle vicinanze influiscono su ogni punto della curva. La curva è rappresentata matamaticamente da una polinomiale di grado uguale a $n - 1$ dove n è l'ordine della curva.

punti di controllo I punti di controllo determinano la forma della curva. Di solito ogni punto della curva è calcolato sulla base di una somma pesata di un numero predefinito di punti di controllo. Il peso di ogni punto varia sulla base di un parametro che lo governa. Il peso di un punto di controllo è un valore diverso da 0 compreso in un intervallo nello spazio del parametro. Dentro questo intervallo, il peso cambia in concordanza con una funzione polinomiale, chiamata *funzione di base*, di un certo grado. Ai limiti dell'intervallo la funzione di base tende a zero con una certa ripidità, tale ripidità viene determinata dal grado del polinomio.

Il fatto che un singolo punto di controllo influenzi solo l'intervallo in cui è attivo è una proprietà molto desiderabile, conosciuta come *supporto locale*. Nella modellazione, questo permette di cambiare parti di una curva mantenendo uguale tutto il resto della stessa.

I punti di controlli possono essere di ogni dimensione. Ad esempio, punti di controllo tridimensionali sono usati abbondantemente nella modellazione 3D.

vettore dei nodi : Il vettore dei nodi è una sequenza di parametri che determina dove e come i punti di controllo influiscono sulla curva NURBS. Il numero di nodi è sempre uguale a $n + d + 1$, dove n è il numero dei punti di controllo e d è il grado della curva. Il vettore dei nodi divide lo spazio parametrico negli intervalli menzionati precedentemente, di solito chiamata *ampiezza del nodo*. Ogni volta che il valore del parametro entra nello spazio di un nuovo nodo, un nuovo

punto di controllo diventa attivo, mentre il vecchio punto di controllo viene scartato. Ne segue che i valori all'interno del vettore dei nodi deve essere ordinato in modo ascendente values in the knot, quindi $(0, 0, 1, 2, 3, 3)$ è un vettore dei nodi valido, mentre $(0, 0, 2, 1, 3, 3)$ non lo è.
Nodi consecutivi possono avere lo stesso valore. Questo implica un'ampiezza del nodo pari a 0, da cui deriva che due punti di controllo sono attivati simultaneamente. Questo influisce sulla contiunuità della curva risultante o sulla sua derivabilità; altresì, ciò permette di creare angoli in una curva altrimenti smussata. Il numero di nodi coincidenti viene spesso chiamato nodo con una certa *molteplicità*. Nodi con molteplicità 2 o 3 sono conosciuti come doppi o tripli nodi. La molteplicità di un nodo è limitata dal grado della curva; poichè una molteplicità più alta spaccherebbe la curva in parti distinte e non userebbe alcuni punti di controllo. Per NURBS di primo grado, quindi, ogni nodo è accoppiato con un punto di controllo.
Il vettore dei nodi di solito inizia con un nodo che ha una molteplicità eguale al grado della curva. Questo ha senso in quanto vengono attivati i punti di controllo che influenzano la prima ampiezza del nodo. Similmente, il vettore dei nodi di solito finisce con un nodo con la stessa molteplicità. Le curve con tale vettore dei nodi iniziano e finiscono in un punto di controllo.
Necessari solo per i calcoli interni, i nodi sono di solito non sono di aiuto agli utenti di software di modellazione. Infatti, in molte applicazioni di modellazione i nodi non vengono resi nè modificabili nè visibili. Di solito è possibile stabilire un vettore di nodi ragionevole osservando le variazioni nei punti di controllo.

Dato il carattere razionale delle NURBS, a differenza delle curve Bézier, è possibile disegnare cerchi perfetti o altre forme altrettanto utili definendo semplicemente la posizione e il peso di pochi punti nello spazio.

2.2 Metodi di input per ambienti 3d

Con l'avvento nell'ultimo decennio di tecnologie in grado di gestire complesse strutture o ambienti 3D, si è reso necessario ricercare nuovi sistemi di controllo per navigare nello spazio 3D. Per poter modificare agevolmente oggetti nello spazio 3D è necessario che il sistema di input abbia 6 gradi di libertà (DOF, *Degrees Of Freedom* in inglese) tre per muoversi nello spazio, x,y e z e tre per gestire la rotazione 3d, α, β e γ.

In [4] *Shumin Zhai*, ricercatore dell'IBM, descrive brevemente alcune idee alla base di alcuni sistemi di input a 6 DOF. Sebbene non si sia ancora stabilito un sistema standard, come il mouse per muoversi nel 2d, lo sviluppo di sistemi di input per ambienti 3D rappresenta un ambito di ricerca prospero, in quanto fino ad oggi sono state proposte molteplici soluzioni al problema.

Zhai nel suo articolo elenca sei aspetti per descrivere le prestazioni di un 6 DOF:

- Velocità
- Precisione
- Facilità di apprendimento
- Fatica nell'utilizzo
- Coordinazione
- Persistenza e acquisizione

Con coordinazione si intende la possibilità di muovere agevolmente tutti e sei i gradi di libertà, mentre con persistenza e acquisizione si intende rispettivamente la possibilità di mantenere il sistema di input nell'ultima posizione raggiunta senza doverlo mantenere fisso manualmente, e per acquisizione si intende la facilità con cui il sistema di input acquisisce i dati dal dispositivo usato.

Descrivendoli con gli attributi elencati, Zhai elenca alcune classi di periferiche di input:

6 DOF basati sul Mouse : si tratta di periferiche che riprendono la forma e la struttura di un mouse e vi aggiungono gradi di libertà. Ciò viene fatto aggiungendo rotelle assieme ai comuni pulsanti del Mouse.

Flying Mice : si tratta di dispositivi che tengono traccia di un puntatore 3d nello spazio. Possono essere ruotati e spostati nell'aria, in modo da garantire i 6 gradi di libertà di cui si ha bisogno. Esistono strumenti di ogni forma appartenenti a questa categoria, dai guanti che percepiscono la posizione della mano, a strumenti simili a mouse, fin'anche dispositivi a forma sferica.

Strumenti da tavolo : vi è poi una schiera di strumenti che consentono agli utenti di muoversi negli ambienti 3d utilizzato uno strumento appoggiato su una piattaforma fissa. Alcuni strumenti di questa classe

permettono, ad esempio, di mouversi nello spazio ruotando una sfera e premendo alcuni pulsanti.

Armature multi-DOF : si tratta di armature meccaniche che come pupazzi simulano l'oggetto da modellare. La più versatile di tali tecnologie è un braccio meccanico che contenendo al suo interno 6 gradi di libertà permette di navigare in ambienti 3d muovendo la mano del braccio robotico nello spazio.

L'ultima classe di strumenti presentata da Zhai contiene al suo interno le cosidette *Tangible Interfaces*, che consistono in strumenti creati ad hoc per alcuni specifici utilizzi che si basano sull'idea di modellare in 3d manipolando un oggetto reale. Esempio di questa classe di periferiche di input viene presentato in [5], dove viene presentato un sistema di manipolazione 3D di un personaggio muovendo un pupazzo dotato di appositi sensori. Questo permette di governare i movimenti 3d di un oggetto virtuale semplicemente muovendo una marionetta, si veda figura 2.4.

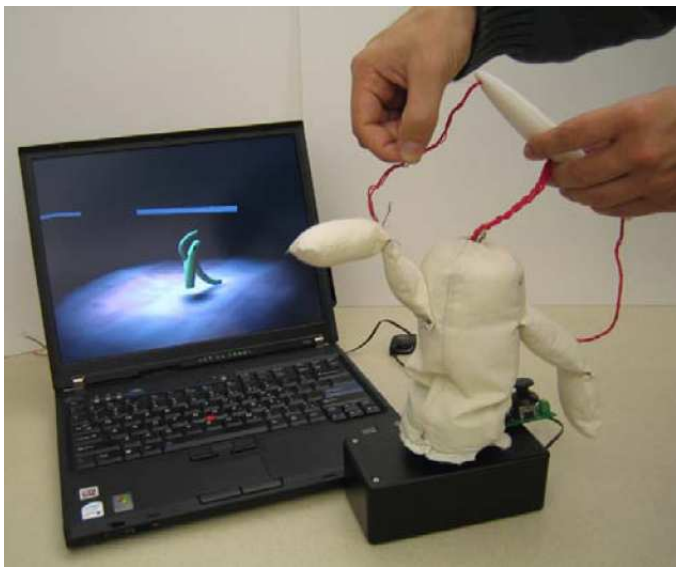
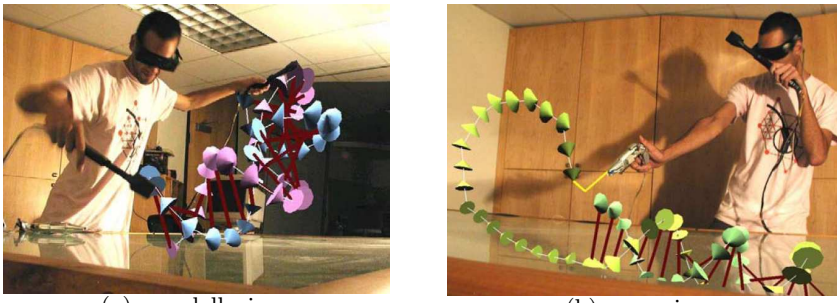


Figura 2.4: esempio di tangible interface

In [6] e in [7] vengono introdotte vie alternative di input grazie all'utilizzo della realtà virtuale. Entrambi rappresentano una soluzione per un singolo predefinito problema.

Il primo articolo, [6], propone uno strumento per manipolare curve e superfici attraverso la realtà virtuale. Ciò viene fatto permettendo all'utente di visualizzare l'ambiente 3d attraverso occhiali speciali che permettono la visione stereoscopica dell'ambiente 3d. L'utente può poi modificare la curva utilizzando dei guanti che riconosciuti da sensori esterni, permettono di manipolare la curva come se si trattasse di punti fisici, tutto attraverso gesti manuali predefiniti.

Il secondo articolo, [7], invece, propone uno strumento per disegnare nello spazio molecole di DNA. Per farlo si serve di occhiali che proiettano sulle lenti immagini stereoscopiche e di particolari strumenti che permettono di manipolare e creare le molecole. Questi strumenti sfruttano sensori integrati che permettono di definirne la posizione. In figura 2.5 è visualizzabile il funzionamento del sistema (Le immagini delle molecole rappresentano ciò che l'utente vede attraverso gli occhiali).



(a) modellazione (b) creazione

Figura 2.5: esempio di input basato su realtà virtuale

Simile per ideazione ai due esempi precedenti, in [8] viene proposto un semplice sistema per la modellazione di oggetti 3D con tecnologie semplici. Questo sistema utilizza semplici occhiali 3d e fascette bianche da mettere sopra le dita, in modo da poter riconoscere la postura della mano semplicemente visualizzando la mano tramite una videocamera, in figura 2.6 è visualizzato un esempio di utilizzo del metodo. Vengono così identificate precise configurazioni delle dita della mano che configurano ognuna un comando per gestire la posizione e la rotazione dei punti che compongono gli oggetti 3d da modellare. La visualizzazione è affidata agli occhiali polarizzati che permettono di rendere le immagini proiettate a schermo tridimensionali.

In [23], libro sulla programmazione grafica 3d in java, vi è un'intero capitolo dedicato a strumenti di input non standard. Il più interessante e

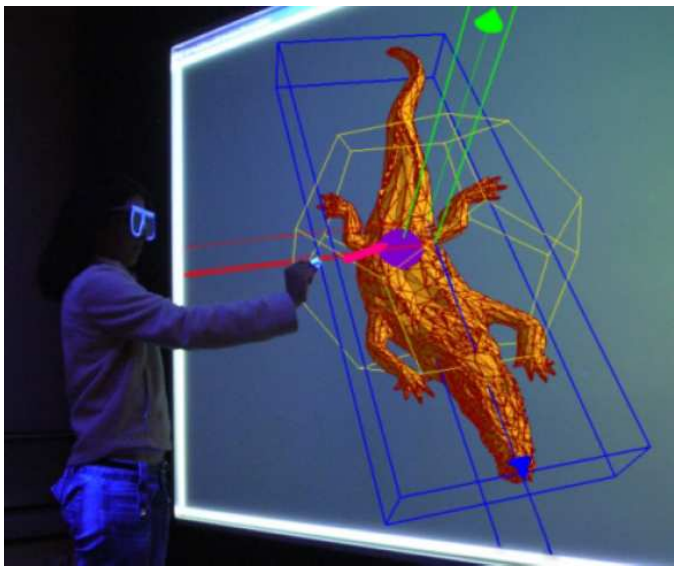


Figura 2.6: esempio di interfaccia

semplice è rappresentato da una fascia di carta avvolgibile al polso disegnata con tre rettangoli di tre colori in netto contrasto, blu, giallo e rosso. Questi rettangoli posti uno sotto l'altro vengono identificati da una webcam, che, riconoscendo il colore predominante, è in grado di definire la posizone spaziale del braccio dell'utente e quindi il comando da attuare nello spazio 3d.

In [9] alcuni ricercatori propongono un sistema di input completamente diverso basato su comandi vocali. Infatti, assieme ai soliti gradi di libertà offerti dai dispositivi standard per interfacce 2d, vengono aggiunti gradi di libertà tramite appositi comandi vocali.

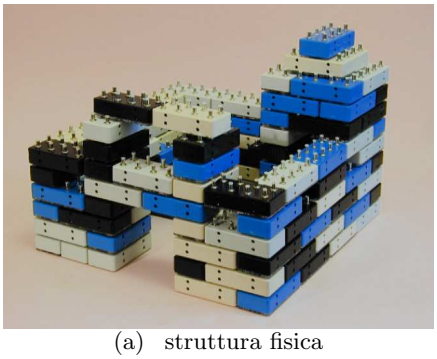
Un altro esempio di *tangible interface* viene offerto in [10] e in [11] dove alcuni ricercatori tedeschi propongono una periferica di input particolare per giochi interattivi per bambini. Questa periferica a forma di cubo, permette di selezionare una opzione tra tante, disponendo la faccia del cubo che identifica l'opzione voluta verso l'alto. In [10] viene visualizzata a schermo una storia interattiva, tramite il cubo è possibile selezionare un'opzione tra tante e controllare l'evolversi della storia. In [11], invece, lo schermo è integrato in ogni faccia del cubo, mentre il principio è lo stesso, controllare l'evoluzione di una storia girando semplicemente il cubo.

In [12] viene descritto un sistema di input manipolabile per navigare o manipolare un ambiente geografico 3d chiamato metaDESK, il tutto funziona utilizzando una cartina geografica di una città, un monitor per visualizzare la ricostruzione 3D e alcuni strumenti fisici, quali una lente, alcuni modellini di edifici... Tramite questi strumenti è possibile decidere quali par-

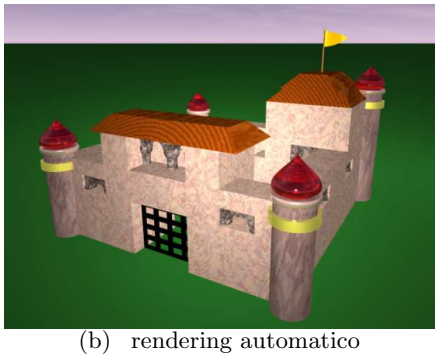
ti della città visualizzare in 3d sul monitor, e modificare gli edifici presenti nell'immagine 3d della città. In figura 2.7 viene visualizzato il metaDESK.

Figura 2.7: metaDESK, esempio di tangile input

Interessante esempio di *tangible interface* è offerto in [13]. In questo articolo viene descritto un sistema per la modellazione di edifici basato su strutture molto simili ai Lego. Questi mattoncini contengono al loro interno alcuni sensori che permettono di inviare al computer centrale le informazioni sulla loro costituzione, individuando così la forma globale definita dall'insieme di mattoncini montati tra loro. Successivamente viene effettuato un processo di auto-Rendering aggiungendo automaticamente particolari alla scena, vedi figura 2.8.



(a) struttura fisica



(b) rendering automatico

Figura 2.8: esempio di tangible interface basato su mattoncini

In [13] viene inoltre proposto un'altro sistema basato sull'idea di tangible interface utilizzando della plastilina. Prima di tutto si modella manualmente l'oggetto che si vuole ricostruire. Successivamente si scansisce l'oggetto con un laser e se ne individua la forma. Infine si ricostruisce in 3D la forma individuata. In figura 2.9 è mostrato tutto il materiale necessario.

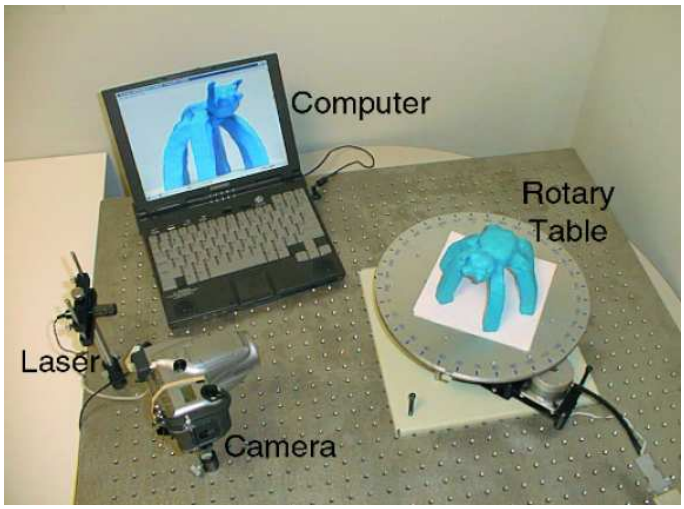


Figura 2.9: esempio di ricostruzione con plastilina

2.3 Ricostruzione 3d

Per procedere all'analisi e quindi all'interpretazione di scene bidimensionali, scene, cioè, in cui esiste un unico punto di vista significativo, è sufficiente la descrizione dell'immagine in termini di contorni e regioni.

Se però la scena ha nette caratteristiche tridimensionali, non basta adottare semplicemente le tecniche di estrazione dei contorni e delle regioni dell'immagine per interpretare correttamente la scena, bensì sono necessarie elaborazioni successive, che mirino a ricostruire, anche se non completamente, la struttura tridimensionale o *shape* (forma) della scena, a partire da una o più immagini. Questa difficolta è insita nel problema perchè nella trasposizione della realtà tridimensionale in immagini bidimensionali si perde una dimensione spaziale: la profondità.

Le tecniche per la ricostruzione della *shape* sono dipendenti dall'osservatore e dal suo particolare punto di vista e permettono di ottenere una descrizione della scena che viene chiamata *2.5Dsketch*. Questo nome è stato scelto proprio per indicare che la rappresentazione non è completa, ma solo,

come si dice nel linguaggio tecnico *ad alto rilievo*.

Esistono cinque principali tecniche di riconoscimento:

Shape from contour La ricostruzione tridimensionale e l'interpretazione di una scena può essere fatta utilizzando semplicemente il disegno "a linee"della stessa scena, senza alcuna informazione pittorica, usando cioè i contorni bidimensionali ottenuti sul piano immagine per proiezione prospettica.

Gli approcci adottati per ricostruire scene tridimensionali da informazioni bidimensionali, utilizzano in modo massiccio, la conoscenza a priori disponibile sulla scena e sono, di conseguenza, quasi simili al processo di interpretazione.

E' stata sviluppata una teoria matematica che va sotto il nome di *inversione prospettica* per affrontare il problema della ricostruzione tridimensionale.

Un altro approccio è la ricostruzione tramite i contorni *line drawing*, ossia da disegni degli spigoli fisici degli oggetti. Limitandosi a scene costituite solo di oggetti con facce piane (il mondo a blocchi), è possibile estrarre informazioni geometriche, in quanto è possibile "etichettare"in modo opportuno le linee in corrispondenza dei tre tipi di contorni tridimensionali: concavi, convessi, di occlusione.

Per informazioni più dettagliate sulla tecnica *shape from contour* si veda [17]. In [18] viene proposto un esempio di utilizzo della tecnica *shape from contour* per il riconoscimento e la ricostruzioni di superfici Gaussiane.

Shape from texture Dal punto di vista concettuale, la ricostruzione tridimensionale basata sulla texture, è analoga alla ricostruzione attraverso i contorni. Per le texture regolari, le metodologie utilizzate per la ricostruzione, si basano sulla considerazione che l'orientamento e la forma della superficie, a cui la texture appartiene, possono essere ricavate analizzando la deformazione geometrica che gli elementi della texture, detti *texel*, hanno subito per prospettiva.

L'analisi delle texture non regolari è molto più complessa. Per queste non è possibile fornire una descrizione in termini geometrici; i texel vengono descritti secondo parametri qualitativi quali il colore, le dimensioni e il valore di intensità luminosa. Le tecniche di ricostruzione, in questo caso, sono basate soprattutto su considerazione statistiche globali.

Shape from shading L'ombreggiatura è un elemento molto importante di

una scena, in quanto favorisce la sua interpretazione tridimensionale, determinando la sensazione di profondità. La tecnica di *shape from shading* parte dall'immagine descritta in termini di livelli di intensità luminosa e utilizza questi ultimi per ricavare una rappresentazione delle superfici presenti sulla stessa scena. Le variazioni di intensità luminosa sono proporzionali all'angolo di orientazione della superficie. Le metodologie di ricostruzione basate sull'ombreggiatura hanno un valore storico in quanto, non richiedendo un processo di elaborazione dell'immagine sono state sviluppattate agli inizi della computer vision. L'utilizzo di questa metodologia è particolarmente importante quando si tratta di esaminare la scena dal punto di vista della luminosità, come nel caso di immagini la cui superficie è completamente priva di elementi pittorici. Per maggiori informazioni sulla tecnica *shape from shading* si veda [15], o [16].

Shape from stereo La tecnica di ricostruzione stereoscopica è quella che più si avvicina alla visione degli esseri umani. E' infatti noto che la profondità di una scena ci risulta evidente in quanto i nostri occhi vedono da punti di vista leggermente diversi. Gli occhi sono leggermente convergenti e i loro assi visivi si incontrano in un punto. La proiezione di questo punto su ognuna delle due retine definisce i *centri di visione* delle stesse. Per cui, qualunque altro punto appartenente al campo visivo, viene proiettato su ogni retina a una certa distanza dal centro di visione. La differenza di questa distanza per i due occhi viene detta *disparità binoculare*, ed è legata alla profondità del punto osservato. Il processo di ricostruzione stereoscopica nella computer vision consiste nel ricavare la tridimensionalità della scena, a partire dalle disparità binoculare. Si possono individuare quattro fasi del processo, di cui le prime due *indentificazione dei tratti* e *determinazione delle corrispondenze*, costituiscono i veri punti problematici della visione stereoscopica; le altre due fasi possono essere pensate come semplici calcoli geometrici.

Shape from motion Un'altra tecnica di ricostruzione è quella che ha a disposizione una telecamera che si muove rispetto alla scena inquadrata. Durante il movimento, le proiezioni dei punti della scena sul piano dell'immagine, si muovono descrivendo curve che vengono indicate come *curve di flusso ottico*. Analizzando queste curve è possibile determinare le distanze dei punti della scena dalla telecamera. Tali distanze possono essere utilizzate per costruire il modello tridimensionale della

scena. Se la telecamera si muove verso la scena con un movimento di traslazione, le curve del flusso ottico saranno rette convergenti in un punto, che viene detto *fuoco di espansione* o, *fuoco di contrazione*, se il sensore si allontana dalla scena. Facendo fare alla telecamera un movimento traslativo, durante il quale vengono acquisite un insieme di immagini della scena, se si conosce il fuoco di espansione del movimento e se tutti i movimenti della telecamera sono noti, è possibile ricostruire la distanza tra i punti della scena e la telecamera stessa. Altre informazioni disponibili in [19].

Le informazioni contenute in questa sezione sono state tratte da [14].

2.4 Conclusioni del capitolo

In questo capitolo si è introdotto il lettore alle tematiche affrontate nell'arco di questa tesi: la rappresentazione matematica delle curve, sistemi di input per ambienti 3d e teorie alla base della ricostruzione 3d grazie alla computer vision.

É opportuno ora identificare brevemente la sottoarea in cui questo lavoro viene svolto. In questa sede si utilizzano le curve Bézier per la rappresentazione grafica delle curve e la teoria delle spline per la ricostruzione 3d. Il software sviluppato rappresenta un'esempio perfetto di *tangible interface*, in quanto partendo da un tubicino fisico si è in grado di ricostruire una curva 3d virtuale. Infine la tecnica di ricostruzione utilizzata appartiene alla classe *shape from contour* da singola immagine.

Nel prossimo capitolo verranno discusse le motivazioni che hanno portato a sviluppare il software e verranno descritti gli strumenti utilizzati, focalizzando per ogni strumento le caratteristiche che hanno contribuito a sceglierle per il progetto rispetto ad altri strumenti affini.

Capitolo 3

Impostazione del problema di ricerca

Nel presente capitolo vengono discusse le motivazioni che hanno portato alla creazione del progetto, come esso è stato sviluppato nel tempo e quali scelte sono state attuate in merito alle tecnologie da utilizzare.

Questo lavoro nasce dalla necessità di ottenere un sistema di input manipolabile per la creazione di curve in un ambiente di sviluppo tridimensionale.

Creare curve in ambiente 3d risulta un compito piuttosto difficoltoso con il solo utilizzo di mouse e tastiera. Per creare una curva bisogna non solo selezionare punto per punto dove si vuole che la curva passi, ma anche predisporre le tangenti in quei punti. Se la curva è abbastanza complicata, l'operazione sopra descritta può risultare lunga e snervante.

Perchè non trovare dunque un modo per costruire materialmente una curva piegando un tubo, e trasferire in qualche modo questa curva in un'ambiente 3d?
Il presente progetto propone una soluzione funzionante del problema.

Nello specifico è stato possibile ottenere un sistema di acquisizione di curve 3d con l'utilizzo di tecnologie alla portata di tutti.

3.1 Descrizione del progetto

In questa sezione viene presentato il progetto realizzato come una black-box. Si forniscono indicazioni su come il sistema può essere utilizzato e su quali strumenti esso utilizzi, sia dal lato fisico, sia dal lato software.

3.1.1 Strumenti fisici utilizzati

Materialmente per questo progetto ci si è serviti di:

- **computer**: ovviamente è necessario un computer per far funzionare un software; da notare che è necessario l'utilizzo di porte USB 2.0.

- **fotocamera digitale**: pultroppo il software funziona solo con alcune tipologie di fotocamera predefinite, le Canon compatibili con le librerie SDK[1]. Per la realizzazione dell'applicativo è stata, infatti, utilizzata una libreria per Matlab compatibile solo con questo tipo di fotocamere, per conoscere quali sono le fotocamere compatibili si veda [20].
  Per lo sviluppo del programma è stata utilizzata la fotocamera Canon A60.

- **tubo**: come tubo è utile usare qualcosa che possa mantenere la forma che gli viene imposta, avendo, ad esempio, un'anima in fil di ferro. Tale tubo non deve essere troppo sottile e deve avere un diametro costante e regolare. Nello specifico il diametro del tubo nella fotografia deve essere di almeno 10-15 pixel ed ogni sezione del tubo deve essere uguale alle precedenti ed alle successive. Al contrario, non esiste alcun vincolo sulla lunghezza del tubo.
  Per lo sviluppo del progetto si è utilizzato un tubicino bianco di diametro 4mm con un'anima in fil di ferro, in grado di mantenere egregiamente la forma.

- **treppiede**: per ottenere una fotografia adeguata alle esigenze è caldamente consigliato l'utilizzo di un treppiede. In tal modo si può fissare e regolare l'inquadratura della fotocamera. Inoltre, l'utilizzo del treppiede permette di tenere ferma l'immagine.

- **pannello**: per ottenere una fotografia ad elevato contrasto è necessario avere un'ambiente uniforme ed opaco, di un colore in forte contrasto con il tubo. Questo può essere ottenuto grazie ad un pannello di colore uniforme e non lucido; può essere utilizzato a questo scopo cartoncino, legno, plastica opaca...
  Per lo sviluppo del software sono stati adoperati fogli da disegno neri.

Utilizzando questi pochi strumenti, vedi immagine 3.1, è possibile creare l'ambientazione giusta per lo scatto della fotografia.

[1] Software Development Key, sono particolari driver che permettono di utilizzare le fotocamere in programmi

Figura 3.1: strumenti fisici utilizzati

Dopo aver montato la fotocamera sul trepiede, si inquadra il tubo posto sopra il pannello, si collega la fotocamera alla porta USB del computer e l'ambientazione è pronta, vedi immagine 3.2.

Un ruolo importante nello scatto della fotografia viene ricoperto dall'illuminazione dell'ambiente. La luce non deve essere diretta per evitare che si formino ombre del tubo sul pannello. Altresì è preferibile usare un'illuminazione diffusa e il più possibile uniforme.

A questo punto si può ottenere la curva desiderata a schermo avviando il software realizzato e impostando i parametri della curva, quali angoli di partenza, numero di punti... Da notare che in primo luogo la fotocamera deve essere accesa.

3.1.2 Strumenti software utilizzati

Per lo sviluppo del programma ci si è serviti di due importanti strumenti software: Matlab e Blender.

Matlab è un software edito dalla **The MathWorks**, per la guida ufficiale si rimanda a [22]. Matlab consiste in un linguaggio di programmazione matematico interpretato, con annesso ambiente di sviluppo; nella

Figura 3.2: ambientazione per lo scatto via remoto

fattispecie Matlab è una piattaforma su cui programmare codice contenente espressioni matematiche complesse. La console di comando Matlab permette di ottenere una risposta immediata alle righe di codice inserite, questa caratteristica permette altresì di utilizzare Matlab come calcolatrice per operazioni complesse.
Il software nasce dall'esigenza di manipolare le matrici, oggetti matematici informaticamente pesanti da trattare. Essendo un linguaggio di alto livello, tale applicazione risulta in grado di effettuare operazioni anche molto complesse scrivendo poche righe di codice. Questa caratteristica è proprio la più apprezzata nell'utilizzo di Matlab.
Nel corso degli anni, Matlab ha aggiunto numerose funzionalità al suo pacchetto software, aumentando notevolmente le sue capacità. Utilizzando tale programma, si è ora si è in grado di visualizzare efficacemente curve e superfici tridimensionali attraverso un motore grafico integrato; si possono inoltre usare funzioni integrate molto complesse e specifiche riferite a ogni campo di ricerca matematica dalle logiche fuzzy alle reti neurali, da simulazioni biologiche a simulazioni finanziarie...
Inoltre il software della The MathWorks è in grado di gestire funzioni create dall'utente attraverso gli M-Files. Questi ultimi sono file conte-

nenti porzioni di codice da richiamare nella consolle di Matlab secondo le necessità.

Blender è un software open-source per la creazione e la gestione di strutture e ambienti tridimensionali, per tutta la documentazione relativa si rimanda a [21]. Blender contiene al suo interno una piattaforma di creazione completamente integrata, in grado di offrire una vasta gamma di strumenti essenziali per la creazione di contenuti 3D; tra questi risaltano la modellazione, l'animazione, il rendering, la post produzione video e la creazione di giochi. La sua interfaccia grafica si basa sulle OpenGL ed è un software multipiattaforma, cioè è in grado di funzionare su diversi sistemi operativi.
Storicamente il progetto Blender nasce nel 1995 dal suo creatore *Ton Roosendaal*, co-fondatore della studio di animazione olandese *NeoGeo*. Nel 1998, Roosendaal decise di fondare una nuova azienda chiamata Not a Number (NaN) per lo sviluppo di Blender. Nel cuore della NaN c'era il desiderio di creare e distribuire gratuitamente una piattaforma di creazione 3D compatta e multi-piattaforma. Dopo un notevole successo iniziale, la NaN non è riuscita a far fronte alle esigenze del mercato, tanto che nel 2001 i finanziatori hanno ritirato il loro appoggio. Ma Roosendaal non si da per vinto e nel 2002 crea l'organizzazione non-profit *Blender Foundation*. Dopo varie controversie, legate ai diritti economici sui sorgenti del programma che hanno portato a una raccolta fondi per 100.000 euro, nel novembre del 2002 Blender è stato rilasciato al mondo nei termini della GNU General Public License (GPL)[2]. Da allora Blender è stato corretto e ampliato da una comunità di 250.000 persone in tutto il mondo, ovviamente sempre sotto la guida dello stesso Ton Roosendaal.
Particolarità di Blender è la sua estendibilità attraverso script Python. Esso infatti fornisce una serie di API per il linguaggio di programmazione Python che permettono di sviluppare applicazioni facenti uso del motore grafico e degli strumenti di Blender. Il software possiede inoltre un motore fisico, chiamato *Game Engine*, che permette lo sviluppo di videogiochi e applicazioni real-time più o meno complesse.

Questi due strumenti, Blender e Matlab, sono stati utilizzati per gestire l'intero software in questa sede realizzato. Matlab si occupa di gestire gli

[2]la licenza GPL definisce i cosidetti software "liberi". Questa licenza permette all'utente finale di usare, modificare, copiare, ridistribuire il software senza limite alcuno. Tramite la GPL l'utente finale oltre al pacchetto software riceve anche il codice sorgente dello stesso.

aspetti matematici del problema, quali l'analisi dell'immagine per la ricostruzione della curva e le rototraslazioni utili a gestire la stessa. Blender si occupa, invece, di gestire l'interfacciamento con l'utente e la visualizzazione in ambiente 3d della curva ricostruita.

Da questa differenziazione di ruoli emerge un problema che si è dovuto affrontare nella sua interezza: il coordinamento tra i due software. Questo è stato possibile grazie al sistema di scripting Python di Blender, che ha permesso di riondurre il problema del coordinamento tra Matlab e Blender ad un problema di coordinamento tra due linguaggi di programmazione, Python e Matlab; questo tipo di problematica si è rivelato molto più risolvibile rispetto al precedente.

3.2 Motivazioni della struttura del progetto

Si è finora compreso da cosa il software è composto e cosa esso realizza, ma quali sono stati i passaggi nel suo sviluppo? Perchè si è scelto di usare necessariamente Matlab e Blender?

In questa sezione si cerca di rispondere a queste domande, esaminando le problematiche riscontrate durante tutte le fasi di sviluppo del lavoro.

Il punto di partenza del progetto è stato l'algoritmo di ricostruzione di curve tridimensionali partendo da singola immagine, sviluppato nell'ambito della tesi di laurea magistrale del Dott. Alessandro Giusti, vedi [25]. Questo algoritmo è disponibile solamente in formato Matlab, in quanto usufruisce di sofisticate librerie per il trattamento delle immagini. Da qui la necessità di usare Matlab all'interno del progetto. Tuttavia, il solo Matlab non era sufficente a garantire l'usabilità del prodotto o soddisfacente una visualizzazione dell'oggetto tridimensionale. Da questa esigenza nasce il lavoro qui trattato.

A questo punto si paravano dinnanzi tante possibilità per raggiungere l'obiettivo prefissato: un sistema funzionante in grado di visualizzare e utilizzare appropriatamente la curva ricostruita, dove tale curva fosse utile.

- Una delle possibilità prevedeva la traduzione dell'algoritmo in altri linguaggi, ad esempio C o Java; questa possibilità però comportava una fatica inutile, in quanto l'algoritmo presentato risultava efficente scritto in Matlab, e una traduzione in un linguaggio come Java lo avrebbe solo rallentato.

- Un'altra possibilità prevedeva lo sviluppo di un software Java sovrastante l'algoritmo Matlab, in grado di comunicare con esso e di visua-

lizzare la curva con gli strumenti di Java 3D o JOGL[3]. Questa opzione avrebbe permesso di creare un'applicazione stand-alone, cioè un piccolo applicativo che non venisse racchiuso in un software più ampio e potente. Ma, in luce di un progetto di tesi di laurea triennale, un'applicazione stand-alone sarebbe risultata carente di possibili sviluppi, nonchè inutile dal punto di vista pratico. Gli unici utilizzi possibili del software sarebbero stati:

  - usarlo come esempio dell'algoritmo di ricostruzione utilizzato, a dimostrazione delle potenzialità di quest'ultimo per sviluppi futuri;
  - usarlo a scopo ludico: creare un'applicazione che usando l'algoritmo di ricostruzione permettesse all'utente di "giocare"in qualche modo.

  Questa alternativa, sebbene più interessante della prima, risultava carente di quell'utilità pratica che si cercava.

- Ultima possibilità era quella di inglobare l'algoritmo in un programma di modellazione 3d già esistente, in modo da permettere all'utente del software di avere uno strumento in più per modellare ambienti tridimensionali. Questa alternativa permetteva di ottenere, non un software completo, ma un'applicativo funzionale a un certo software di largo utilizzo. Sebbene questa caratteristica possa sembrare riduttiva rispetto a un'applicazione stand-alone, in quanto l'applicazione risulterebbe legata alle sorti del software per cui è stata sviluppata, questa stessa caratteristica rende il lavoro svolto più potente e, sopratutto, più utile. Grazie a questa caratteristica, infatti, il software prodotto è in grado di sfruttare le potenzialità del software per cui è stato sviluppato, permettendo allo sviluppatore di "creare un castello senza doversi curare delle fondamenta". Ciò è proprio cosa succede nel presente caso: utilizzando un programma di modellazione 3d non ci si deve preoccupare della visualizzazione degli oggetti 3d creati o della loro manipolazione, bensì basta riuscire a trasformare il formato della curva restituito dall'algoritmo Matlab con il formato necessario al software di modellazione e a quel punto usare gli strumenti messi a dispo-

[3]Java3D e JOGL sono due librerie Java per la gestione della grafica tridimensionale all'interno di programmi Java. Le due librerie si basano entrambe sulle OpenGL, librerie grafiche del C, ma solo JOGL ne rappresenta l'effettiva traduzione. Per maggiori informazioni [23].

sizione da quest'ultimo per generare ulteriori sviluppi dell'applicativo prodotto.

Come si evince dalla descrizione del progetto, l'alternativa di sviluppo affrontata è stata proprio la terza, dove come software di modellazione 3d si è scelto Blender.

Ma perchè proprio Blender?
Le motivazioni sono facilmente intuibili data la descrizione precedente del software. Si è scelto Blender principalmente per due motivi:

1. Blender è un software Open-Source, condizione questa che permette di usufruire del programma attraverso la licenza GPL precedentemente descritta. Questa caratteristica permette di ottenere il software senza dover pagare esosi diritti, i software di modellazione tridimensionale in genere sono molto costosi, ma soprattutto di accedere ai sorgenti e di modificare il codice all'occorrenza. Ciò implica la possibilità di sviluppare software sopra Blender, semplicemente rispettando le condizioni della licenza GPL, che impone che tutto il software prodotto sia sotto licenza GPL.

2. Blender permette di essere esteso: si possono programmare applicazioni che facciano uso di Blender e che ne estendano le funzionalità. Blender permette altresì di essere esteso in due modi:

   (a) tramite script Python: si tratta del metodo più utilizzato, basato su API scritte in Python che Blender fornisce per interagire con tutti gli elementi presenti nel programma;

   (b) tramite plugin Binari: consiste in programmi C che, diversamente dal caso precedente, lavorano direttamente sul programma principale di Blender; infatti, se il plugin contiene un errore e termina, termina anche lo stesso Blender.

   Grazie a questi due metodi è possibile avere pieno controllo degli oggetti presenti all'interno di Blender, modificandole a piacere con procedure automatiche.

Risulta ora chiaro perchè si è scelto Blender a discapito di un qualsiasi altro programma di modellazione tridimensionale.

Una volta scelto Blender si è reso necessario scegliere il modo con cui esso sarebbe stato esteso. La scelta è ricaduta subito sul sistema di script Python, percepito come metodo più semplice per manipolare Blender. Per una breve descrizione del linguaggio di programmazione Python si rimanda

alla sezione 3.3.

Scelti gli strumenti software da utilizzare, il primo problema da affrontare è stato il trasferimento dei dati della curva da Matlab allo script Python, interfaccia di Blender. Per coordinare i due linguaggi di programmazione si è innanzitutto pensato di far comunicare i due processi tramite librerie di sistema operativo Python, gestendo la comunicazione tramite i pid[4]. Questa soluzione risultò fallimentare per alcune stranezze riscontrate nell'avvio di Matlab: esso non si avviava direttamente, ma veniva richiamato da un processo padre che terminava immediatamente dopo l'avvio effettivo di Matlab. In secondo luogo si è pensato di gestire la comunicazione attraverso file di comunicazione esterni. Questo metodo ha sì trovato una soluzione, però concettualmente poco pulita, richiedendo l'intervento del file-system per la comunicazione tra i due processi.
Infine, si è trovata la soluzione al problema del coordinamento utilizzando una libreria *wrapper*[5] tra Matlab e Python. La libreria, sviluppata per ambiente Linux, è stata, non senza problemi, adattata al sistema operativo Windows, ambiente dove il software è stato sviluppato. Una volta resa funzionante la libreria, la comunicazione tra i due processi ha cessato di essere un problema, in quanto è stato possibile richiamare le funzioni Matlab necessarie all'applicativo da sviluppare direttamente dallo script Matlab.

Gestita la comunicazione tra i due processi, si è proceduto direttamente al disegno della curva in Blender, sebbene si sia dovuto provvedere alle opportune traduzioni di formato della curva nella funzione Matlab che gestisce la ricostruzione. Gli aspetti matematici di traduzione verranno discussi nei prossimi capitoli, poichè in questa sede è sufficente comprendere che è stato necessario affrontare il problema del formato della curva, differente nei due software, Matlab e Blender.

Può essere utile a questo punto introdurre brevemente cosa si intende per curva in Blender, fonte [21]. In Blender vi sono due modi per rappresentare le curve:

- **Curve Bézier**: sono composte da numerosi punti di controllo consistenti in un punto e due maniglie. Il punto, nel mezzo, è usato per spostare e gestire l'intero punto di controllo; le maniglie, invece, consentono di modificare la forma della curva. Una curva di Bézier è tangente al segmento che passa per il punto e la maniglia. La "ripi-

[4] *process identifier* identifica un valore intero che nell'ambito di sistema operativo indica un processo in esecuzione sulla macchina.

[5] letteralmente involucro, si intende con wrapper una libreria che traduce le istruzioni da un linguaggio ad un'altro, incapsulando le istruzioni dell'uno in istruzioni dell'altro.

dità"della curva è controllata altresì dalla lunghezza della maniglia. Esistono quattro tipi di maniglie:

- **Maniglia Libera**: può essere usata in qualsiasi modo si vuole.
- **Maniglia Allineata**: giace sempre su una linea retta.
- **Maniglia Vettore**: entrambe le parti di questa puntano alla maniglia precedente o alla successiva.
- **Maniglia Automatica**: questa maniglia ha una lunghezza ed una direzione completamente automatiche, impostate da Blender per avere il risultato il più levigato possibile.

Da notare che matematicamente le curve di Bézier rappresentano funzioni cubiche, cioè polinomiali di terzo grado.

- **Curve NURBS**: letteralmente Non Uniform Rational B-Splines, sono definite come polinomiali razionali che interpolano dei punti di controllo chiamati *nodi*. La curva ha diverse proprietà che la definiscono; nello specifico tre:
  - la prima proprietà viene fissata sui nodi iniziale e finale attraverso una pre-impostazione:
    - se la pre-impostazione vale **Uniform**: la curva non passa nè per il nodo iniziale nè per quello finale, ma interpola con la polinomiale anch'essi;
    - se, altrimenti, la pre-impostazione vale **Endpoint**: sia il nodo iniziale e che quello finale appartengono alla curva stessa.
  - la seconda proprietà riguarda l'ordine della polinomiale che approssima i punti. Dipendentemente dalla variabile **Order**, la "profondità"del calcolo della curva varia: se l'ordine vale 1 la curva è un punto, se vale 2 la curva è lineare, se vale 3 la curva è quadratica, e così via fino ad un massimo di 6 per cui la curva è di quinto grado. Parlando matematicamente, **Order** è sia l'ordine del Numeratore sia del Denominatore della polinomiale razionale che definisce le NURBS.
  - infine, ad ogni nodo è assegnato un "peso", *Weight*, che definisce la proporzione con cui un nodo partecipa alla "deformazione"della curva. Varia da un minimo di 0 per cui la curva non risente in alcun modo del nodo selezionato, a un massimo di 10 valore per cui la curva passa per il nodo selezionato.

Una volta definite le due tipologie di rappresentazione delle curve, è doveroso spiegare perchè si è scelta una forma a discapito dell'altra. La scelta è ricaduta sulle curve Bézier in quanto perfettamente compatibili con le curve generate da Matlab, ma anche perchè decisamente più semplici da gestire rispetto alle NURBS. Infatti, se è possibile con le Bézier avere diretto controllo dei punti che apparterranno alla curva, ciò non fattibile per le NURBS con le quali, invece, risulta non immediato definire i punti per i quali passerà la curva.

Giunti a questo punto, le esigenze iniziali del lavoro, cioè ottenere in Blender la curva elaborata da Matlab, sono state soddisfatte; è ora possibile ricercare nuove funzionalità per l'applicativo prodotto, occupandosi in primis della costruzione di strutture a più curve, ma anche della gestione dell'orientamento delle curve stesse.

Per occuparsi di questo aspetto si è pensato di utilizzare la matematica delle matrici di rototraslazione; tramite alcuni passaggi è stato possibile ottenere il completo controllo della curva attraverso tre angoli e un punto di origine. Questo sistema, adeguatamente corretto, ha permesso di ricavare un'ulteriore funzionalità: l'innesto di una curva su di un punto di un'altra curva. Ciò è stato consentito dalla possibilità di estrapolare il sistema di riferimento (s.d.r.) nel punto di interesse della curva Bézier; grazie a questo, infatti, è stato possibile prendere come s.d.r. iniziale della nuova curva l's.d.r. della curva originale nel punto di innesto e, da quì, impostare la nuova curva con gli angoli voluti.

Infine, l'ultima cosa da fare risulta essere questa: trovare un modo per interfacciarsi con una fotocamera digitale, in modo da poter ottenere le fotografie del tubo direttamente da remoto. É stato possibile ovviare efficacemente a questo problema attraverso le librerie di interfacciamento della Canon chiamate SDK; è stata trovata, infatti, una libreria per Matlab che, usando le SDK, permette di effettuare scatti fotografici da remoto tramite una porta USB. Questa libreria, sebbene limiti l'utilizzo della funzionalità di scatto da remoto alle sole fotocamere Canon compatibili, rappresenta la migliore alternativa possibile in quanto solo alcuni modelli di fotocamere digitali permettono lo scatto da remoto; oltretutto, in genere questa funzionalità viene permessa tramite driver proprietari, difficilmente utilizzabili nell'ambito di un software estraneo.

Risolto anche il problema dello scatto da remoto, il software realizzato è risultato pittosto completo, in grado di soddisfare le esigenze alle quali ci si era prefissati di rispondere.

3.3 Caratteristiche e funzionalità di Python

Risulta opportuno in questa sede esporre una breve introduzione al linguaggio di programmazione Python. Tutte le informazioni relative a Python possono essere trovate in [29].

Il linguaggio Python nasce ad Amsterdam nel 1989, dove il suo creatore *Guido Van Rossum* lavorava come ricercatore.

3.3.1 L'interprete Python

Python è un linguaggio di script pseudocompilato. Questo significa che ogni programma sorgente deve essere pseudocompilato da un interprete. L'interprete è un normale programma che va installato sulla propria macchina, che si occuperà prima di interpretare il codice sorgente e successivamente di eseguirlo. Il principale vantaggio di questo sistema risulta essere la portabilità: lo stesso programma potrà girare su una piattaforma Linux, Mac o Windows purché vi sia installato l'interprete.

L'interprete Python supporta anche un modo d'uso interattivo attraverso il quale è possibile inserire codice direttamente da un terminale, visualizzando immediatamente il risultato. Questo risulta certamente un bel vantaggio per chi sta imparando il linguaggio, ma anche per gli sviluppatori esperti, in quanto brevi tratti di codice possono essere provati in modo interattivo prima di essere integrati nel programma principale.

3.3.2 Struttura del linguaggio

Python è un linguaggio multi-paradigma. Infatti, esso permette in modo agevole di scrivere programmi seguendo il paradigma *object oriented*, la programmazione strutturata o la programmazione funzionale.

Il controllo dei tipi è forte (*strong typing*) e viene eseguito runtime (*dynamic typing*). In altre parole una variabile può assumere nella sua storia valori di tipo diverso, pur appartenendo in ogni istante ad un tipo ben definito.

Python utilizza un *garbage collector*[6] per la gestione automatica della memoria.

[6] per garbage collector si intende un sistema automatico che permette di cancellare le parti di memoria non più indirizzabili da una variabile nel programma.

3.3.3 Tipi base, tipi contenitore e classi

Python possiede un gran numero di tipi base. Oltre ai tipi interi e floating point classici, supporta in modo trasparente numeri interi arbitrariamente grandi e numeri complessi. Supporta altresì tutte le operazioni classiche sulle stringhe con una eccezione: le stringhe in Python sono oggetti immutabili, cosicché qualsiasi operazione che in qualche modo potrebbe alterare una stringa (come ad esempio la sostituzione di un carattere) restituirà invece una nuova stringa.

Avendo Python una tipizzazione dinamica, tutte le sue variabili sono in realtà semplici puntatori ad oggetto (*reference*): sono gli oggetti invece ad essere dotati di tipo. Ad esempio, ad una variabile cui è stato assegnato un intero, può successivamente essere assegnata una stringa o un array.

In Python c'è un moderato controllo dei tipi a runtime. Si ha conversione implicita per i tipi numerici, per cui si può ad esempio moltiplicare un numero complesso per un intero, pur non essendoci, ad esempio, conversione implicita tra numeri e stringhe, per cui un numero è un argomento non valido per le operazioni su stringa.

Python ha una serie di tipi contenitori come ad esempio liste, tuple e dizionari. Questi tipi contenitori sono sequenze e, in quanto tali, condividono la maggior parte dei metodi. Le liste sono array estendibili, invece le tuple sono array immutabili di lunghezza prefissata. Gli elementi dei dizionari, invece, sono composti da una coppia di dati separati da due punti ":". Il primo elemento della coppia rappresenta l' indice (detto "chiave"); il secondo, invece, è il suo valore corrispondente. Ogni elemento è detto coppia chiave-valore.

Python permette l'utilizzo delle classi, essendo predisposto per la programmazione object oriented, e ammette l'ereditarietà multipla.

3.3.4 Costrutti per il controllo di flusso

Python ha solo due forme di ciclo:

- **for** che cicla sugli elementi di una lista o su di un iteratore
- **while** che cicla fin tanto che l'espressione booleana indicata risulterà vera.

Allo stesso modo Python ha solamente il costrutto **if**...**elif**...**else** per le scelte condizionate.

Una cosa inusuale del linguaggio Python è il metodo che usa per delimitare i blocchi di programma, che lo rende unico fra tutti i linguaggi più

diffusi. Python, infatti, invece di usare parentesi o parole chiavi, usa l'indentazione stessa[7] per indicare i blocchi nidificati.

3.3.5 Programmazione funzionale e gestione delle eccezioni

Un'altro punto di forza in Python è la disponibilità di elementi che facilitano la programmazione funzionale. Infatti, Python permette di avere funzioni come argomenti.

Python supporta ed usa estensivamente la gestione delle eccezioni come mezzo per controllare la presenza di eventuali condizioni di errore.

Le eccezioni permettono il controllo degli errori più conciso ed affidabile rispetto a molti altri modi possibili, usati in genere per segnalare errori o situazioni anomale. Le eccezioni sono thread-safe: non sovraccaricano il codice sorgente, e inoltre possono facilmente propagarsi verso l'alto nello stack delle chiamate a funzione quando un errore deve venire segnalato ad un livello più alto del programma.

Il modo di fare frequente consiste, invece che nel fare controlli preventivi, nell'eseguire direttamente l'azione desiderata, catturando invece le eventuali eccezioni che si verificassero.

3.3.6 Libreria standard

Python ha una vasta libreria standard, il che lo rende adatto a molti impieghi. Oltre ai moduli della libreria standard se ne possono aggiungere altri scritti in C oppure Python per soddisfare le proprie esigenze particolari.

La libreria standard è uno dei punti forti di Python. Essa infatti è compatibile con tutte le piattaforme, ad eccezione di poche funzioni, segnalate chiaramente nella documentazione come specifiche di una piattaforma particolare. Grazie a questo generalmente anche programmi Python molto grossi possono funzionare su Linux, Mac, Microsoft Windows e altre piattaforme senza dover essere modificati.

3.3.7 Prestazioni del linguaggio

Se paragonato ai linguaggi compilati statically typed, come ad esempio il C, la velocità di esecuzione non è uno dei punti di forza di Python, specie nel

[7] si può usare sia una tabulazione, sia n spazi bianchi, ma lo standard Python è 4 spazi bianchi

calcolo matematico. Tuttavia, Python permette di aggirare in modo facile l'ostacolo delle performance pure: è infatti relativamente semplice scrivere un'estensione in C o C++ e poi utilizzarla all'interno di Python, sfruttando così l'elevata velocità di un linguaggio compilato solo nelle parti in cui effettivamente serve, e sfruttando invece la potenza e versatilità di Python per tutto il resto del software.

Proprio quest'ultimo fattore riguardante le prestazioni è ciò che ha favorito la decisione di mantenere l'algoritmo di ricostruzione della curva in Matlab. Infatti, sebbene probabilmente sarebbe stato possibile riscrivere l'algoritmo in Python, esso non sarebbe stato efficente quanto l'algoritmo scritto in Matlab, mentre ricercando la comunicazione tra i due linguaggi è possibile mantenere le prestazioni di Matlab all'interno di un programma Python.

3.4 Conclusioni del capitolo

In questo capitolo sono stati mostrati i problemi affrontati nello sviluppo del progetto. Si è discusso inoltre delle tecnologie e degli strumenti utilizzati, argomentando le motivazioni che hanno portato al loro utilizzo.

Nel prossimo capitolo si vedranno più nel dettaglio gli aspetti logici e matematici che stanno alla base del lavoro svolto. Si descriverà la struttura del progetto senza parlare dell'effettiva implementazione. Per l'implementazione si rimanda al capitolo 5.

Capitolo 4

Progetto logico della soluzione del problema

Questo capitolo tratta degli aspetti logico matematici del problema affrontato. Vengono focalizzati i vari punti in cui il problema è composto e come essi sono stati risolti.

Strutturalmente il lavoro risulta suddiviso in 5 parti, di cui una ulteriormente scomponibile in 3 sottoparti:

1. acquisizione in remoto dell'immagine da fotocamera digitale
2. elaborazione dell'immagine in Matlab
   (a) calcolo dei punti della curva nello spazio 3d
   (b) trasformazione della curva da spline a Bézier
   (c) rototraslazione della curva secondo angoli predefiniti
3. trasferimento dati da Matlab a blender
4. elaborazione e visualizzazione curva in blender
5. interfaccia utente in blender

In seguito verranno affrontati punto per punto gli aspetti elencati; si rimandano al prossimo capitolo i dettagli sull'implementazione di quest'ultimi.

4.1 Acquisizione da fotocamera in remoto

Per riuscire ad effettuare lo scatto via remoto si è ricorso a una libreria per Matlab: camerabox [24].

CameraBox è un toolbox [1] per Matlab che permette di comunicare via USB con una fotocamera digitale Canon, lasciando trasparire all' utente comandi della fotocamera (scatto, flash,...) come se fossero semplici funzioni Matlab.

Il toolbox fa riferimento a librerie proprietarie della canon: le CSDK (Canon Software Development Key), scritte in C. Ciò permette a CameraBox di comunicare con i driver della fotocamera, permettendo di acquisire informazioni sulle impostazioni della stessa, e di modificarle secondo le possibili opzioni: si può quindi facilmente selezionare i parametri voluti, ad esempio flash o apertura focale, con cui scattare successivamente la fotografia, tutto sempre in remoto e automatizzabile programmando codice Matlab.

Grazie a questo toolbox si è potuto automatizzare il processo di acquisizione dell'immagine da cui ricostruire la curva tridimensionale: da Blender viene chiamato Matlab attraverso il procedimento descritto in 4.3, che tramite il toolbox attiva la fotocamera, scatta la fotografia e la salva sull' hard disk secondo un path predefinito, rendendola raggiungibile dallo script Matlab che si occuperà di elaborare l'immagine.

4.2 Elaborazione dell'immagine in Matlab

Matlab permette di salvare i programmi scritti con il linguaggio di programmazione Matlab in particolari file, chiamati M-files. Questi si dividono in due tipologie:

script : righe di codice eseguibile raggruppate in un file;

funzioni : funzioni richiamabili dalla console di comando di Matlab, o da altri script, con valori di input e output.

L'M-file le cui parti verrano descritte in questa sezione è una funzione. Come tale ha diversi parametri in input:

- il path del file immagine da cui ricostruire la curva
- il numero di punti che si vuole compongano la curva
- il punto nello spazio 3d da cui ha origine la curva
- tre angoli α β γ che identifichino la direzione di partenza della curva

[1] letteralmente s̈catola degli strumentiïn Matlab definisce una libreria, un insieme di funzioni, che permettono a Matlab di ampliare le sue capacità

- due vettori che definiscono il piano di riferimento iniziale in cui inserire la curva

La funzione restituisce in output i punti della curva Bézier associata all'immagine secondo gli angoli definiti.

Nelle successive sottosezioni si vedrà nel dettaglio come matematicamente la funzione lavora.

4.2.1 Ricostruzione della curva 3d

L'algoritmo di ricostruzione di una curva a partire da una singola immagine è stato sviluppato nell'ambito della tesi di laurea di Alessandro Giusti [25] e rappresenta il punto di inizio di questo lavoro.

Il metodo si basa sulla geometria delle *canal surfaces*[2], e sfrutta l'effetto prospettico intrinseco nella fotografia.

L'algoritmo in questione appartiene alla classe di problemi denominata *shape from contour*, cioè a quelli che usano il contorno dell'oggetto nella fotografia per ricostruire la tridimensionalità dello stesso. Il metodo trova una soluzione solo se si conoscono le proprietà del modello dell'oggetto da osservare e se il sistema possiede una certa regolarità, caratteristica che si riscontra nelle *canal surfaces*.

L'estrazione del contorno della *canal surface*, nel caso qui esposto un tubo, è il primo passo della ricostruzione 3d della curva; per questa fase è stato utilizzato l'algoritmo di Canny-Deriche.

Successivamente si ricercano i punti accoppiati sui due contorni trovati precedentemente. I punti accoppiati sono le immagini di due punti sulla *canal surface* appartenenti alla stessa sezione. Questo procedimento è descritto in [25].

Una volta trovati sufficenti punti accoppiati, che identificano univocamente una sezione del tubo, i punti appartenenti al centro di ogni sezione formano i punti della curva cercata. La terza dimensione mancante viene calcolata a partire dalla differenza di raggio nelle sezioni, effetto questo dovuto alla prospettiva.

Infine questi punti si interpolano in una spline, definita in 2.1.1. La spline ottenuta è una funzione costituita da tanti *breaks*[3] quanti sono i punti calcolati precedentemente e poi interpolati. Ogni break è collegato al successivo da una funzione di terzo grado rendendo così continua la curva identificata

[2] canal surfaces sono dei cilindri generalizzati a sezione circolare costante ottenuti come inviluppo di sfere di raggio R costante, il cui centro si muove su un asse curvilineo tale che il raggio di curvatura sia sempre superiore a R.

[3] i breaks in una spline sono tutti i punti in cui è suddivisa la curva

dalla spline.

Si è così riuscito a ottenere una curva in un formato matematico preciso che nella prossima sezione verrà elaborato per ottenere una curva nel formato compatibile con Blender.

4.2.2 Trasformazione della Spline in formato Bézier

A questo punto dello script, la curva è identificata da una spline che, come abbiamo descritto nella sezione precedente, è una serie di punti detti *breaks* collegati fra loro da polinomiali cubiche definite da quattro coefficenti. L' equazione che la descrive è:

$$y = a(x - \text{break}_i)^3 + b(x - \text{break}_i)^2 + c(x - \text{break}_i) + d \qquad (4.1)$$

dove:

$$x \in [\text{break}_i, \text{break}_{i+1}] \qquad \forall i$$

L'obbiettivo è trasformare l'informazione contenuta nella spline in informazione compatibile con Blender, cioè in una curva Bézier,vedi 2.1.2.

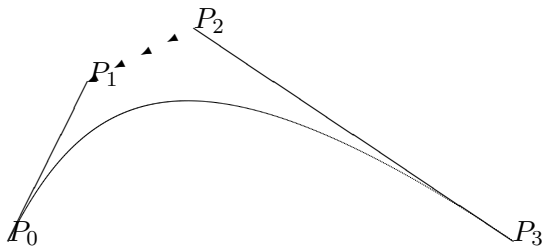


Figura 4.1: esempio di curva Bézier

Una curva Bézier passante per i punti P_0,con tangente destra P_1, e per P_3,con tangente sinistra P_2,(vedi figura 4.1) è definita dall'equazione:

$$f(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3 \quad t \in [0,1] \qquad (4.2)$$

La curva Bézier vera e propria è definita da $p-1$ equazioni del tipo 4.2 dove p è il numero di punti che la descrivono.

L'idea a questo punto è di far coincidere le due curve in modo da avere la trasformazione desiderata. Bisogna far si che i punti P_i siano in funzione con i coefficenti nei vari breaks. Per fare ciò il primo passo sarà trasformare la x in funzione di t ed andarla a sostituire nell'equazione 4.1:

$$t = \frac{x - \text{break}_i}{\text{break}_{i+1} - \text{break}_i} \quad x, t \in [0, 1]$$

dove la x è stata divisa per $\text{break}_{i+1} - \text{break}_i$ per far coincidere i due domini,quindi:

$$x = t \cdot (\text{break}_{i+1} - \text{break}_i) + \text{break}_i \tag{4.3}$$

e sostituendo in 4.1 si ottiene:

$$\begin{aligned} y = a(t \cdot (\text{break}_{i+1} - \text{break}_i) + \text{break}_i - \text{break}_i)^3 + \\ +b(t \cdot (\text{break}_{i+1} - \text{break}_i) + \text{break}_i - \text{break}_i)^2 + \\ +c(t \cdot (\text{break}_{i+1} - \text{break}_i) + \text{break}_i - \text{break}_i) + d \end{aligned}$$

da cui:

$$y = a(\text{break}_{i+1} - \text{break}_i)^3 t^3 + b(\text{break}_{i+1} - \text{break}_i)^2 t^2 + c(\text{break}_{i+1} - \text{break}_i)t + d \tag{4.4}$$

dove:

$$t \in [0, 1]$$

Giunti a questo punto si espande l'equazione della curva Bézier,la 4.2:

$$f(t) = P_0(1 + 3t^2 - 3t - t^3) + 3P_1(t + t^3 - 2t^2) + 3P_2(t^2 - t^3) + P_3 t^3 \tag{4.5}$$

da cui riportandola in forma polinomiale:

$$f(t) = (P_3 - 3P_2 + 3P_1 - P_0)t^3 + (3P_2 - 6P_1 + 3P_0)t^2 + (3P_1 - 3P_0)t + P_0 \tag{4.6}$$

Come ultimo passaggio bisogna ora imporre l'uguaglianza tra la polinomiale Bézier(4.6) e l'equazione della spline in t (4.4),dove a $\text{break}_{i+1} - \text{break}_i$ si sostituisce Δbreak_i:

$$(P_3 - 3P_2 + 3P_1 - P_0)t^3 + (3P_2 - 6P_1 + 3P_0)t^2 + (3P_1 - 3P_0)t + P_0 = \\ = a\Delta\text{break}_i^3 t^3 + b\Delta\text{break}_i^2 t^2 + c\Delta\text{break}_i t + d$$

la cui unica soluzione è definita dal sistema:

$$\begin{cases} a\Delta\text{break}_i^3 = P_3 - 3P_2 + 3P_1 - P_0 \\ b\Delta\text{break}_i^2 = 3P_2 - 6P_1 + 3P_0 \\ c\Delta\text{break}_i = 3P_1 - 3P_0 \\ d = P_0 \end{cases} \tag{4.7}$$

che risolta in funzione di a,b,c e d ottiene:

$$\begin{cases} P_0 = d \\ P_1 = \frac{1}{3}c\Delta\text{break}_i + d \\ P_2 = \frac{1}{3}b\Delta\text{break}_i^2 + \frac{2}{3}c\Delta\text{break}_i + d \\ P_3 = a\Delta\text{break}_i^3 + b\Delta\text{break}_i^2 + c\Delta\text{break}_i + d \end{cases} \tag{4.8}$$

che rappresenta proprio ciò che si stava cercando.

Si è così giunti a una trasformazione, sistema 4.8, in grado di cambiare la forma della curva da Spline a Bézier mantenendo gli stessi punti di definizione; bisogna ricordare che la curva Bézier è caratterizzata da 3 punti spaziali per ogni punto che la definisce: tangente sinistra, punto stesso, tangente destra.

L'ultimo punto da affrontare per modificare le coordinate della curva è impostare il punto di partenza e la direzione di inizio. Infatti, il sistema di riferimento intrinseco nella curva è basato sul centro ottico della fotocamera, coerentemente con l'algoritmo descritto in [25].

4.2.3 Rototraslazione della curva

Per ottenere la direzione desiderata si è fatto uso di rototraslazioni consecutive del sistema di riferimento della curva; queste verranno descritte qui di seguito.

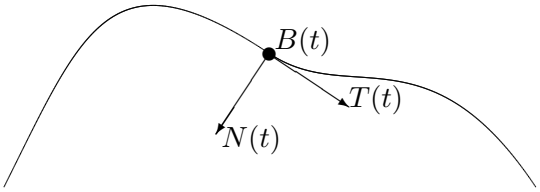


Figura 4.2: esempio di TNB frame con binormale uscente dal foglio

Sistema di riferimento iniziale

Innanzitutto si trasla la curva in modo da far coincidere il suo primo punto con l'origine degli assi, punto $(0, 0, 0)$.

In seguito si cambia il sistema di riferimento nella curva secondo i due vettori, tangente e binormale, in input alla funzione, vedi pag. 43, in modo da adeguarsi al sistema dove la curva avrà origine nella scena di Blender; se ne vedrà l'utilità nella sezione 4.4. I vettori in input sono nella forma:

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \\ 0 \end{bmatrix} \quad B = \begin{bmatrix} B_x \\ B_y \\ B_z \\ 0 \end{bmatrix} \tag{4.9}$$

dove l'ultimo valore, lo 0, nella matematica delle coordinate omogenee rende il vettore una direzione.

Per completare il sistema di riferimento bisogna trovare l'ultimo vettore ortogonale agli altri due, la normale. Seguendo la matematica delle TNB frame [4], vedi [26] e figura 4.2, per trovare la normale basta fare il prodotto vettoriale tra B e T:

$$N(t) = B(t) \times T(t) \tag{4.10}$$

che in forma matriciale si esprime:

[4]Dato un punto di una curva nello spazio, il vettore tangente(T),il vettore normale(N) e il vettore binormale(B) sono tre vettori ortogonali di lunghezza unitaria che insieme formano un sistema di coordinate naturale a quel punto, questo sistema è il TNB frame

$$N = \begin{bmatrix} 0 & -B_z & B_y & 0 \\ B_z & 0 & -B_x & 0 \\ -B_y & B_x & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} T_x \\ T_y \\ T_z \\ 0 \end{bmatrix} \tag{4.11}$$

dove si è omesso il tempo in quanto i tre vettori si intendono presi nell'origine a $t = 0$. N calcolato in questo modo è nella forma:

$$N = \begin{bmatrix} N_x \\ N_y \\ N_z \\ 0 \end{bmatrix} = \begin{bmatrix} B_y T_z - B_z T_y \\ B_z T_x - B_x T_z \\ B_x T_y - B_y T_x \\ 0 \end{bmatrix} \tag{4.12}$$

Si è dunque ottenuto un nuovo sistema di riferimento basato sui tre assi T,N e B. Questo andrà sostituito al sistema di riferimento iniziale, composto dai versori x,y e z situato nell' origine.

Il centro del nuovo sistema di riferimento è identificato dal punto in input allo script, vedi pag. 42. La nuova origine sarà nella forma:

$$P = \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \tag{4.13}$$

dove il quarto valore uguale a 1 identifica, nell'ambito delle coordinate omogenee, che il vettore è un punto.

A questo punto si deve costruire la matrice di rototraslazione R dove il vettore T sarà sostituito al versore x, N a y e B a z. Imponendo inoltre l'origine in P si avrà:

$$R = \begin{bmatrix} T & N & B & P \end{bmatrix} = \begin{bmatrix} T_x & N_x & B_x & P_x \\ T_y & N_y & B_y & P_y \\ T_z & N_z & B_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

Ricordando che il sistema di riferimento iniziale della curva è identificato dalla matrice identità I, e che $I \cdot A = A$ dove A è una matrice qualsiasi, si ottiene che il sistema di riferimento S della curva è:

$$S = I \cdot R \tag{4.15}$$

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} T_x & N_x & B_x & P_x \\ T_y & N_y & B_y & P_y \\ T_z & N_z & B_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.16}$$

da cui:

$$S = R = \begin{bmatrix} T_x & N_x & B_x & P_x \\ T_y & N_y & B_y & P_y \\ T_z & N_z & B_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.17}$$

Si è così riusciti a posizionare l'origine della curva nel punto e nel sistema di riferimento desiderati.

Rototraslazioni α, β e γ

Ora si deve tener conto dei valori degli angoli in input, α,β e γ in modo da ottenere la direzione desiderata per la curva, rispettando il nuovo sistema di riferimento. Si vuole fare in modo che l'angolo α identifichi la direzione della tangente della curva, nel suo primo punto, rispetto ai nuovi assi X e Y, l'angolo β rispetto agli assi X e Z, mentre l'angolo γ deve rappresentare la direzione della normale alla curva, sempre nell'origine della curva.

Si costruiscono quindi tre matrici di rototraslazione come segue:

$$\rho_\alpha = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.18}$$

che identifica una rotazione di un angolo α rispetto all'asse z;

$$\rho_\beta = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.19}$$

che identifica una rotazione di un angolo β rispetto all'asse y;

$$\rho_\gamma = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma & 0 \\ 0 & \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.20}$$

che identifica una rotazione di un angolo γ rispetto all'asse x.

La rototraslazione totale si trova quindi moltiplicando le tre rotazioni, le formule 4.18, 4.19 e 4.20, facendo attenzione a moltiplicare per l'inversa di ρ_β invece che per ρ_β semplice. Quest'ultimo passaggio serve per ruotare l'asse x verso l'asse z e non viceversa, come accadrebbe invece mantenendo la rotazione semplice per la regola della mano destra[5]. Quindi:

$$\rho = \rho_\alpha \cdot \rho_\beta^{-1} \cdot \rho_\gamma \tag{4.21}$$

cioè:

$$\rho = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma & 0 \\ 0 & \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.22}$$

che risolta:

$$\rho = \begin{bmatrix} \cos\alpha\cos\beta & -\sin\alpha\cos\gamma - \cos\alpha\sin\beta\sin\gamma & \sin\alpha\sin\gamma - \cos\alpha\sin\beta\cos\gamma & 0 \\ \sin\alpha\cos\beta & \cos\alpha\cos\gamma - \sin\alpha\sin\beta\sin\gamma & -\cos\alpha\sin\gamma - \sin\alpha\sin\beta\cos\gamma & 0 \\ \sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.23}$$

Questa matrice rappresenta la rototraslazione rispetto ai tre angoli che si stava cercando. La matrice 4.23, va a moltiplicare la matrice 4.17, componendo così un'ulteriore trasformazione S':

[5] infatti la rotazione rispetto all'asse y segue la regola della mano destra che impone alla rotazione positiva di girare in senso antiorario e quindi dall'asse z all'asse x

$$S' = S \cdot \rho = \begin{bmatrix} T_x & N_x & B_x & P_x \\ T_y & N_y & B_y & P_y \\ T_z & N_z & B_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} c\alpha c\beta & -s\alpha c\gamma - c\alpha s\beta s\gamma & s\alpha s\gamma - c\alpha s\beta c\gamma & 0 \\ s\alpha c\beta & c\alpha c\gamma - s\alpha s\beta s\gamma & -c\alpha s\gamma - s\alpha s\beta c\gamma & 0 \\ s\beta & c\beta s\gamma & c\beta c\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.24}$$

dove c e s stanno rispettivamente per cos e sin.

La matrice 4.24 risolta sarà nella forma:

$$S' = \begin{bmatrix} A'_x & B'_x & C'_x & P_x \\ A'_y & B'_y & C'_y & P_y \\ A'_z & B'_z & C'_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.25}$$

dove:

$$A'_i = T_i c\alpha c\beta + N_i s\alpha c\beta + B_i s\beta \tag{4.26}$$

$$B'_i = T_i(-s\alpha c\gamma - c\alpha s\beta s\gamma) + N_i(c\alpha c\gamma - s\alpha s\beta s\gamma) + B_i c\beta s\gamma \tag{4.27}$$

$$C'_i = T_i(s\alpha s\gamma - c\alpha s\beta c\gamma) + N_i(-c\alpha s\gamma - s\alpha s\beta c\gamma) + B_i c\beta c\gamma \tag{4.28}$$

per $i = x, y, z$.

Correzione del sistema di riferimento intrinseco nella curva

Pultroppo c'è ancora un particolare da sistemare: la curva restituita dall'algoritmo di ricostruzione, spiegato nella sezione 4.2.1, ha intrinscamente una tangente, una normale e una binormale non corrispondenti agli assi naturali x, y, z, ma dipendenti dalla posizione spaziale del tubo rispetto alla macchina fotografica.

Qui di seguito verrà spiegato come si è ovviato a questo problema. Innanzitutto bisogna conoscere i valori di questi 3 vettori. Ciò è possibile ricordando le proprietà non ancora citate delle *TNB frame*, in bibliografia [26]:

$$T_2(t) = \frac{x'(t)}{\|x'(t)\|} = \frac{dx}{ds} \tag{4.29}$$

dove $x'(t)$ rappresenta la derivata prima della funzione definita da 4.1 e $\|x'(t)\|$ rappresenta la norma di $x'(t)$ calcolabile con il teorema di pitagora:

$\sqrt{x_x'^2(t) + x_y'^2(t) + x_z'^2(t)}$.

Il calcolo dei 3 vettori deve essere fatto all'origine della curva per $t = 0$, nella spline nel primo punto di *break*. Risulta quindi immediato:

$$x(0) = a \cdot 0^3 + b \cdot 0^2 + c \cdot 0 + d = d \tag{4.30}$$

$$x'(0) = 3a \cdot 0^2 + 2b \cdot 0 + c = c \tag{4.31}$$

$$x''(0) = 6a \cdot 0 + 2b = 2b \tag{4.32}$$

Bisogna precisare inoltre che in una spline per ogni dimensione vi è una cubica; vi saranno dunque 12 coefficenti per ogni break, 4 per la x, 4 per la y e 4 per la z.

La tangente sarà quindi il vettore:

$$T_2(0) = \begin{bmatrix} \frac{c_x}{\sqrt{c_x^2+c_y^2+c_z^2}} \\ \frac{c_y}{\sqrt{c_x^2+c_y^2+c_z^2}} \\ \frac{c_z}{\sqrt{c_x^2+c_y^2+c_z^2}} \\ 0 \end{bmatrix} \tag{4.33}$$

Rimane ora da ottenere la normale e la binormale. Esiste una formula anche per questo:

$$B_2(t) = \frac{x'(t) \times x''(t)}{\|x'(t) \times x''(t)\|} \tag{4.34}$$

quindi:

$$B_2(0) = \begin{bmatrix} \frac{c_y b_z - c_z b_y}{\sqrt{(c_y b_z - c_z b_y)^2 + (c_z b_x - c_x b_z)^2 + (c_x b_y - c_y b_x)^2}} \\ \frac{c_z b_x - c_x b_z}{\sqrt{(c_y b_z - c_z b_y)^2 + (c_z b_x - c_x b_z)^2 + (c_x b_y - c_y b_x)^2}} \\ \frac{c_x b_y - c_y b_x}{\sqrt{(c_y b_z - c_z b_y)^2 + (c_z b_x - c_x b_z)^2 + (c_x b_y - c_y b_x)^2}} \\ 0 \end{bmatrix} \tag{4.35}$$

è immediato trovare la N dall'equazione 4.12.

$$N_2 = \begin{bmatrix} \frac{(c_z b_x - c_x b_z)\cdot c_z - (c_x b_y - c_y b_x)\cdot c_y}{\sqrt{(c_y b_z - c_z b_y)^2 + (c_z b_x - c_x b_z)^2 + (c_x b_y - c_y b_x)^2}\cdot\sqrt{c_x^2 + c_y^2 + c_z^2}} \\ \frac{(c_x b_y - c_y b_x)\cdot c_x - (c_y b_z - c_z b_y)\cdot c_z}{\sqrt{(c_y b_z - c_z b_y)^2 + (c_z b_x - c_x b_z)^2 + (c_x b_y - c_y b_x)^2}\cdot\sqrt{c_x^2 + c_y^2 + c_z^2}} \\ \frac{(c_y b_z - c_z b_y)\cdot c_y - (c_z b_x - c_x b_z)\cdot c_x}{\sqrt{(c_y b_z - c_z b_y)^2 + (c_z b_x - c_x b_z)^2 + (c_x b_y - c_y b_x)^2}\cdot\sqrt{c_x^2 + c_y^2 + c_z^2}} \\ 0 \end{bmatrix} \tag{4.36}$$

Giunti a questo punto non rimane che costruire l'ultima matrice di rototraslazione R_2 componendo i tre vettori T_2, N_2 e B_2:

$$R_2 = \begin{bmatrix} T_2x & N_2x & B_2x & 0 \\ T_2y & N_2y & B_2y & 0 \\ T_2z & N_2z & B_2z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.37}$$

e moltiplicarla alla matrice 4.25, facendo attenzione a moltiplicare per l'inversa[6]:

$$S_2 = S' \cdot R_2^{-1} = \begin{bmatrix} A'_x & B'_x & C'_x & P_x \\ A'_y & B'_y & C'_y & P_y \\ A'_z & B'_z & C'_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} T_2x & N_2x & B_2x & 0 \\ T_2y & N_2y & B_2y & 0 \\ T_2z & N_2z & B_2z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \tag{4.38}$$

La rototraslazione ultimata sarà nella forma:

$$S_2 = \begin{bmatrix} A_fx & B_fx & C_fx & P_x \\ A_fy & B_fy & C_fy & P_y \\ A_fz & B_fz & C_fz & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.39}$$

dove si lascia il calcolo dei coefficienti A_fi, B_fi, C_fi ai lettori più audaci.

Si è riusciti infine ad ottenere la matrice di rototraslazione finale che permette di avere il completo controllo dell'inizio della curva attraverso un punto e tre angoli.

La funzione Matlab è finita; si passerà ora a descrivere la logica dietro lo script Python che permette l'utilizzo di Blender.

[6]bisogna invertire la matrice per annullare gli effetti iniziali dei tre vettori sulla curva, sottraendo al sistema gli angoli la differenza rispetto agli assi naturali x,y,z

4.3 Comunicazione Matlab-Blender

La comunicazione tra i processi Matlab e Blender è gestita da una apposita libreria per Python: Matlabwrap [27]. Questa è un ponte fra Python e Matlab che permette di rimappare le funzioni Matlab in funzioni Python.

Matlabwrap fa uso di una versione open-source della libreria Matlab chiamata engine, vedi [28]; questa permette la comunicazione tra programmi C e Matlab.

Matlab engine opera in un processo in background, separato dal programma C che lo richiama. Il processo comunica tramite *pipe*[7] in sistemi simil UNIX, tramite la piattaforma *COM* (Component Object Model)[8]in Microsoft Windows. L' engine fornisce poche funzioni, visualizzate in tabella 4.1.

Nome Funzione	**Scopo**
engOpen	Apre la sessione di Matlab engine
engClose	Chiude la sessione
engGetVariable	Ottiene una matrice dall' engine
engPutVariable	Manda una matrice all' engine
engEvalString	Esegue un comando Matlab
engOutputBuffer	Crea un buffer per testo in output da Matlab
engOpenSingleUse	Apre l'engine per un solo comando e termina
engGetVisible	Ritorna true se l'engine è visualizzato
engSetVisible	Mostra o nasconde la shell di Matlab

Tabella 4.1: tabella delle funzioni di Matlab engine

Matlabwrap lavora in modo leggermente differente: maschera al programmatore le funzioni dell'engine di Matlab e fornisce ai programmi Python una libreria fittizia dei comandi Matlab. L'utente Python vede solo un modulo che rappresenta Matlab, chiamato *mlab*, che permette di specificare tutti i comandi semplicemente digitando `mlab.comando`.

In realtà Matlabwrap, dopo aver aperto l'engine all'importazione del modulo, traduce questi comandi nelle funzioni di tabella 4.1; successivamente ottiene informazioni relative ai parametri dei comandi e gestisce input e

[7]le pipe usano due file descriptor per la comunicazione tra processi. Il flusso di dati viene scritto da un processo (**scrittore**) sul file descriptor in scrittura e viene letto da un'altro processo (**lettore**) sul file descriptor in lettura

[8]COM gestisce la comunicazione tra processi attraverso l'interfaccia *IStream* che permette di gestire flussi di dati attraverso i metodi *read* e *write*.

output delle funzioni in maniera trasparente, traducendo questi nel formato numerico di Python, chiamato numpy[9]. Una volta riconosciuto, il comando Matlab viene associato alla stringa mlab.comando; ciò permette alle successive chiamate della stessa funzione di risparmiare tempo.

Questo risolve il problema della comunicazione tra i due script. Lo script Python, tramite Matlabwrap, richiama i due script di Matlab, di cui uno scatta la fotografia e l'altro elabora la curva. Successivamente gli script Matlab terminano e restituiscono le loro elaborazioni al chiamante.

4.4 Disegno curva in Blender

La curva in formato Bézier, una volta ottenuta, viene facilmente disegnata tramite le librerie Python di Blender.

Blender, per ogni punto della curva, necessita delle coordinate della tangente[10] sinistra, le coordinate del punto stesso e le coordinate della tangente destra. Questi punti vengono trasformati in un particolare formato chiamato *BezTriple*. I punti in forma *BezTriple* vengono passati in input alla funzione che crea la curva, alla quale viene aggiungento un punto alla volta. Fatto ciò, si registra la curva alla scena corrente, che immediatamente viene disegnata sulla finestra di blender.

Si vedrà ora nel dettaglio il formato delle informazioni trasferite da Matlab a Blender. I punti che compongono la curva sono incapsulati dentro una struttura matriciale $9 \times n$ (matrice 4.40), dove n indica quanti punti compongono la curva. In questa matrice le prime tre colonne identificano le coordinate delle tangente sinistre, le tre colonne centrali quelle dei punti, e le ultime tre le tangenti destre. .

$$\begin{bmatrix} Tsx_{x1} & Tsx_{y1} & Tsx_{z1} & P_{x1} & P_{y1} & P_{z1} & Tdx_{x1} & Tdx_{y1} & Tdx_{z1} \\ Tsx_{x2} & Tsx_{y2} & Tsx_{z2} & P_{x2} & P_{y2} & P_{z2} & Tdx_{x2} & Tdx_{y2} & Tdx_{z2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ Tsx_{xn} & Tsx_{yn} & Tsx_{zn} & P_{xn} & P_{yn} & P_{zn} & Tdx_{xn} & Tdx_{yn} & Tdx_{zn} \end{bmatrix} \tag{4.40}$$

[9] necessita di un modulo apposito, non rappresenta un tipo build-in

[10] dato il versore che identifica la tangente in un punto della curva e dato un coeffieciente moltiplicativo riferito alla curvatura della curva nello stesso punto, κ, si definiscono coordinate della tangente le tre coordinate spaziali riferite agli assi x,y e z, ottenute sommando le coordinate del punto stesso con il vettore ottenuto moltiplicando il versore per κ.

Data la struttura della matrice 4.40, risulta immediato estrarre le nove coordinate che contraddistinguono una tripla Bézier; infatti, ogni riga rappresenta in questo modo una *BezTriple*. Scandendo riga per riga la matrice e passando le righe alla funzione *BezTriple()*, si ottengono in sequenza tutti i punti da aggiungere alla curva; suddetta curva viene creata al momento dell'inserimento del primo punto.

Una volta ottenuta la curva, vengono apportate le modifiche alla stessa, dipendentemente dai flag settati dall'utente nell'interfaccia grafica. Si può, ad esempio, dare sostanza alla curva disegnando una struttura cilindrica attorno ad essa, ottenendo quindi un tubo, il tutto semplicemente impostando alcuni flags in automatico. Dipendentemente dalle scelte dell'utente si puo quindi richiedere allo script diverse opzioni:

la consistenza a tubo : illustrata nel paragrafo precedente;

la posizione : oltre ad impostare gli angoli ed il punto d'origine si può decidere di mantenere il sistema di riferimento originale;

numero di punti della curva : l'utente può scegliere quanti punti devono comporre la curva, in modo da bilanciare la complessita della stessa con le esigenze dell'utente.

scattare o meno la foto : l'utente decide se scattare da remoto la foto o elaborare un'immagine già presente nell'hard-disk;

usare il flash o meno : dipendentemente dall'illuminazione del setting si può impostare la presenza del flash.

Il software permette, inoltre, di aggiungere curve ad altre curve già esistenti nella scena. Per fare ciò, l'utente deve inizialmente selezionare la curva alla quale aggiungerne un'ulteriore e, successivamente, decidere in quale punto ordinato della stessa debba essere attaccata la nuova curva.

Si può ora comprendere il motivo per cui è necessario lasciare la possibilità di cambiare il sistema di riferimento iniziale, argomrnto trattato nella sezione 4.2.3: per creare una curva collegata a un'altra bisogna conoscere il sistema di riferimento del punto sulla curva, il *TNB frame* vedi [26], che sarà origine della seconda curva. Così facendo i tre angoli di definizione della seconda curva sono relativi alla prima; ciò permette di ottenere curve con una certa continuità. Matematicamente bisogna trovare i tre vettori nel punto sulla curva selezionato, T, N e B. Ricordando l'equazione 4.6 si possono calcolare la derivata prima e seconda:

$$f(0) = P_0 \tag{4.41}$$

$$f'(0) = 3P_1 - 3P_0 \tag{4.42}$$

$$f''(0) = 6P_2 - 12P_1 + 6P_0 \tag{4.43}$$

dove i quattro punti sono:

P_0 : le coordinate del punto selezionato;

P_1 : le coordinate della sua tangente destra;

P_2 : le coordinate della tangente sinistra del punto successivo;

P_3 : le coordinate del punto successivo.

Date le equazioni 4.29 e 4.34 si ha:

$$T = \frac{f'(0)}{\|f'(0)\|} \tag{4.44}$$

$$B = \frac{f'(0) \times f''(0)}{\|f'(0) \times f''\|} \tag{4.45}$$

quindi:

$$T = \begin{bmatrix} \frac{3P_{1x} - 3P_{0x}}{\text{norm}_t} \\ \frac{3P_{1y} - 3P_{0y}}{\text{norm}_T} \\ \frac{3P_{1z} - 3P_{0z}}{\text{norm}_t} \\ 0 \end{bmatrix} \tag{4.46}$$

$$B = \begin{bmatrix} \frac{(3P_{1y} - 3P_{0y}) \cdot (6P_{2z} - 12P_{1z} + 6P_{0z}) - (3P_{1z} - 3P_{0z}) \cdot (6P_{2y} - 12P_{1y} + 6P_{0y})}{\text{norm}_b} \\ \frac{(3P_{1z} - 3P_{0z}) \cdot (6P_{2x} - 12P_{1x} + 6P_{0x}) - (3P_{1x} - 3P_{0x}) \cdot (6P_{2z} - 12P_{1z} + 6P_{0z})}{\text{norm}_b} \\ \frac{(3P_{1x} - 3P_{0x}) \cdot (6P_{2y} - 12P_{1y} + 6P_{0y}) - (3P_{1y} - 3P_{0y}) \cdot (6P_{2x} - 12P_{1x} + 6P_{0x})}{\text{norm}_b} \\ 0 \end{bmatrix} \tag{4.47}$$

in cui norm_t e norm_b sono le norme dei due vettori.

Si hanno così i due vettori, tangente e binormale, da passare in input alla funzione Matlab che calcola la curva. Si otterrà dunque una seconda curva fuoriuscente dalla prima secondo i tre angoli, α, β e γ, selezionati dall'utente.

Ulteriori vantaggi attribuibili alla possibilità di agganciare più curve sono l'inserimento in testa o in coda di un'ulteriore curva. Per ottenere un'inserimento in coda basta selezionare l'ultimo punto della curva ed impostare i valori dei tre angoli a 0, ottenendo così la continuità fino alla derivata seconda. Specularmente, si può inserire in testa selezionando il primo punto e facendo assumere ai tre angoli i valori: $\alpha = 0$, $\beta = 180$ e $\gamma = 0$

4.5 Interfaccia utente in Blender

L'interfaccia grafica di Blender lascia molta libertà all'utente. Si può decidere quali e quante finestre visualizzare in un dato istante. Si possono inoltre tenere attive più di una interfaccia. Si ha così la possibilità di mantenere, ad esempio, quattro finestre 3d, una per ogni visuale, ed una finestra delle opzioni in una singola schermata.

Per le interfacce grafiche degli script Python vi è una struttura di finestre apposita chiamata *Scripts Window*. Questa finestra consiste in uno spazio vuoto riempibile da pulsanti implementati nello script.

Blender lavora con un' interfaccia ad eventi. Dal lato della programmazione si ha quindi la necessità di gestire sia la visualizzazione dei vari pulsanti, sia la gestione degli eventi da essi richiamati. Per fare ciò esistono due moduli appositi in Blender: **Draw** e **BGT**; il primo lavora con oggetti, agganciandosi all'iterfaccia di Blender, il secondo è invece un *wrapper*[11] per le OpenGL. Questo lavoro fa uso soltanto del primo modulo.

Il modulo **Draw** rende disponibili diversi tipi di pulsanti di cui si useranno:

Button: semplici pulsanti che attivano un evento;

Toggle: particolari pulsanti che impostano un flag a true o false;

String: caselle di testo da modificare per salvare stringhe;

Number: casella che permette l'inserimento di numeri; essa può esser modificata anche dalle frecce laterali.

Usando **Draw** si rende necessario scrivere tre differenti funzioni. Nella prima, chiamata, ad esempio, `Gui()`, vanno disegnati i pulsanti. Nella seconda, `Keyevents(evt,val)`, vanno descritti gli eventi, `evt`, riferiti alla pressione, `val` settato a true, o al rilascio, `val` settato a false, di un pulsante della tastiera o del mouse. Nella terza, infine, si inseriranno gli eventi relativi ai pulsanti creati nella funzione `Gui()`; questa viene chiamata `Button(evt)`. Una volta scritte, le tre funzioni si registrano nell'interfaccia di Blender attraverso la funzione `Register(Gui,Keyevents,Button)`; ciò disegna effettivamente l'interfaccia nella scena e la rende permanente in Blender.

[11] wrapper, in questo caso, è un involucro che permette alle informazioni di passare da un formato a un altro: rende quindi compatibili le OpenGL, scritte per programmi c, con Python.

Figura 4.3: interfaccia script

L'interfaccia dello script che si presenta all'utente è visualizzata nella figura 4.3. Vi sono due *Button*, quattro *Toggle* e una casella di testo (*String*). Il primo *Button*, chiamato *Curve*, disegna la curva richiamando gli script, con opzioni dipendenti dai *Toggle*. La casella di testo sottostante, *pd:...*, permette di specificare la path nella quale si vuole vengano salvate le fotografie scattate in remoto.

I quattro *Toggle* identificano rispettivamente:

Photo : la possibilità di fotografare in remoto;

Flash : la possibilità di usare il flash ;

Tube : la possibilità di visualizzare la curva come se fosse un tubo[12];

Orig : la possibilità di avere la curva originale di Matlab.

L'ultimo *Button*, *quit*, termina lo script.

Le funzioni del *Button* chiamato *Curve* variano a seconda del fatto che sia o meno selezionata una curva della scena. Se non viene selezionata alcuna curva, ed il *Toggle Photo* non è impostato, apparirà sulla finestra un menu di selezione di file, vedi figura 4.4.

Dopo aver selezionato l'immagine, o alla pressione di *Curve*, se è impostato il *Toggle Photo*, si rivela un *popup-block*[13], vedi figura 4.5.

[12] questo procedimento viene attuato costruendo automaticamente una struttura poligonale attorno alla curva, ciò si ottiene impostando i valori di estrusione a valori definiti di default

[13] popup-block è una finestra temporanea in primo piano

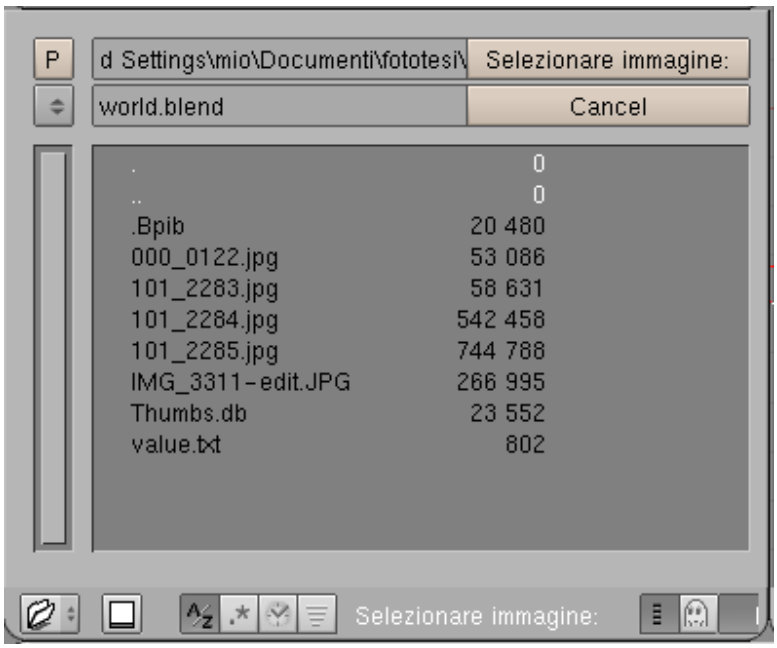


Figura 4.4: finestra di selezione del file

Questo blocco contiene quattro *Number*. Il primo identifica il numero di punti della curva, mentre gli altri tre indicano i tre angoli per la definizione dell'inizio della curva, α, β e γ. Una volta selezionati i valori desiderati, si preme il pulsante laterale con scritto OK per salvare le impostazioni e chiudere il blocco. A questo punto viene avviato lo script Matlab e viene disegnata la curva.

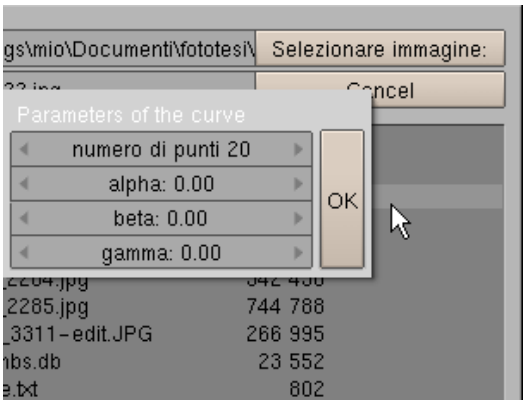


Figura 4.5: Pup block

Se, invece, viene selezionata una curva nella schermata 3D View, appare un popup-block leggermente diverso.

Il blocco in figura 4.7 appare se non vi sono altre curve associate alla curva selezionata. Viceversa, il blocco in figura 4.6 appare se la curva

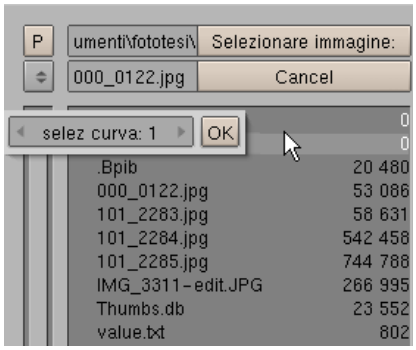


Figura 4.6: pup-block con aggiunta a struttura a più curve

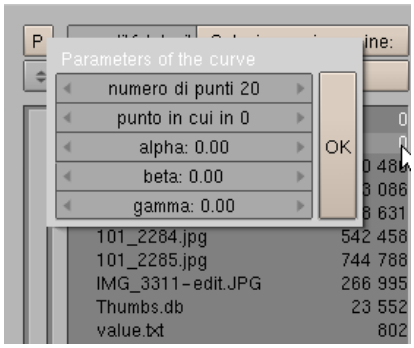


Figura 4.7: pup-block con aggiunta a singola curva

selezionata costituisce una struttura complessa composta da più curve. Il blocco 4.6 contiene una sola opzione; questa identifica la curva a cui si vuole aggiungerne un'altra. Le curve sono ordinate cronologicamente, cioè sono numerate in base alla costruzione della struttura di curve: la curva origine avrà numero 0, la prima curva aggiunta ad essa avrà numero 1, la seconda avrà numero 2, la prima curva aggiunta alla prima curva aggiunta sarà la numero 3, ...

Nel blocco 4.7 c'è un'opzione in più rispetto al blocco 4.5; questa identifica l'ennesimo punto sulla curva al quale si vuole costruire un'altra curva. Questo punto sarà quello in cui si prenderà il sistema di riferimento iniziale della nuova curva.

Finisce qui l'interagibilità dell'interfaccia grafica dello script. Non resta che visualizzare correttamente la curva in Blender, cosa che si fa semplicemente aprendo una o più finestra in modalità *3D View*. Il risultato viene visualizzato in figura 4.8 e ciò chiude questo capitolo.

4.6 Conclusioni del capitolo

In questo capitolo si è parlato del progetto logico del sistema, di come è strutturato e di come si è matematicamente giunti ai risultati spiegati. Si è visto come il sistema sia composto di 5 parti concettuali, affrontate ad una ad una nel corso del capitolo.

Nel prossimo capitolo si tratteranno gli aspetti implementativi, basati sui linguaggi di programmazione Python e Matlab, legati alle parti concettuali fin qui trattate.

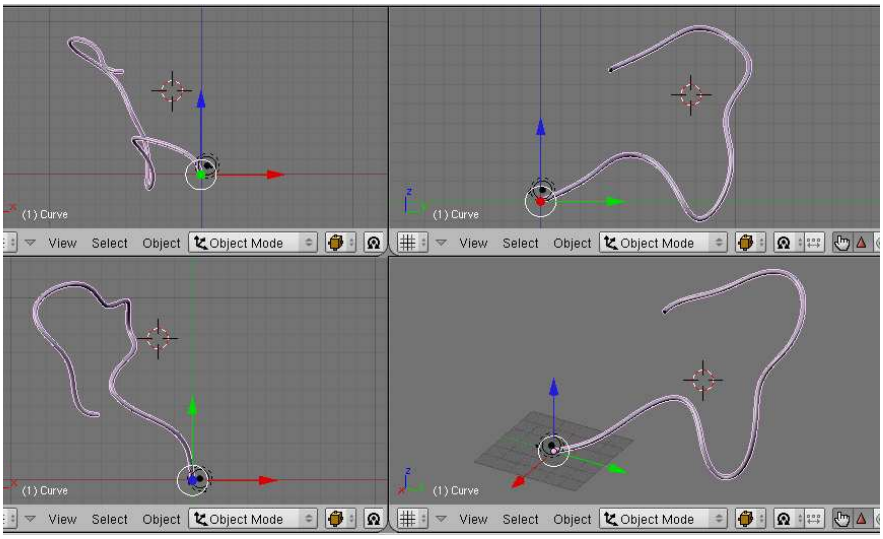


Figura 4.8: visualizzazione della curva 3d in Blender su 4 schermate

Capitolo 5

Aspetti Implementativi

In questo capitolo vengono analizzati gli aspetti implementativi di quanto trattato nel capitolo 4. Si riprendono ad uno ad uno tutte le parti del programma e si scende nel dettaglio delle librerie e degli strumenti utilizzati. Pertanto, si mantiene la struttura pentapartita del capitolo precedente.

Qui di seguito si richiamano brevemente i contenuti delle parti presentate:

Scatto in remoto con CameraBox: per gestire lo scatto in remoto si è utilizzato un toolbox esterno di Matlab; tramite questo strumento è possibile scattare fotografie, con o senza flash, e salvarle direttamente in Hard-Disk.

Elaborazione dell'immagine e ricostruzione della curva: in questa parte si è visto come la curva viene ricostruita dall'immagine ed in seguito modificata per essere compatibile con Blender. Nello specifico la curva viene prima trasformata da Spline a curva Bézier, successivamente rototraslata secondo tre diverse matrici:

la prima corrispondente ad un sistema di riferimento iniziale riferito al setting in cui la curva verrà posta;

la seconda corrispondente a tre rotazioni secondo gli angoli α, β e γ;

la terza deputata ad annullare l'effetto del sistema di riferimento intrinseco nella curva.

Comunicazione Matlab-Blender con Matlabwrap: nel precedente capitolo è stato spiegato il metodo con cui si è ovviato ai problemi di comunicazione tra Matlab e Blender attraverso Matlabwrap. Questa libreria permette a Python di interagire con Matlab come se fosse un

modulo chiamato `mlab` e di comporre i comandi Matlab semplicemente scrivendo `mlab.comando`.

Disegno Curva in Blender: è stato mostrato come la curva viene elaborata all'interno di Blender e come sia possibile creare strutture complesse a più curve.

Interfaccia Grafica di Blender: si è introdotto il modulo `Draw` e si è mostrata l'effettiva interfaccia grafica utilizzata per lo script qui trattato.

Il lettore è ora pronto ad affrontare correttamente i problemi legati alla programmazione, senza doversi più curare degli aspetti logico progettuali già affrontati in precedenza.

In questo capitolo verranno, inoltre, descritti i tentativi svolti prima di arrivare alla soluzione finale.

5.1 Scatto in remoto con CameraBox

CameraBox [24] viene gestita tramite una serie di funzioni. Di seguito vengono descritte quelle usate per il presente lavoro:

`[CameraControlHandle cc] = InitCameraBox`: la funzione inizializza *CameraBox* e restituisce un *handler* [1], `cc`;

`[CameraControlHandle cc] = DestroyCameraBox(cc)`: termina *CameraBox* tramite l'*handler* creato in precedenza, restituisce un *handler* vuoto;

`[bool IsError, string ErrorType] = CB_IsError(cc)`: la funzione controlla se l'*handler* è in stato di errore; se il booleano restituito, `IsError`, vale `True` è avvenuto un errore che viene descritto nella stringa `ErrorType`;

`[cell(string)CameraNames] = CB_GetConnectedCameraNames(cc)`: restituisce una lista con tutti i dispositivi, cioè fotocamere, connessi al computer tramite porte USB;

`CB_OpenSource(cc, index)`: si connette al dispositivo numero `index` della lista restituita da `CB_GetConnectedCameraNames(cc)`;

`CB_CloseSource(cc)`: si disconnette dalla fotocamera corrente;

[1] un handler è una variabile numerica che permette di gestire una funzione complessa, esempio è l'identificativo del socket

`CB_EnterReleaseControl(cc)`: entra nella **modalità scatto** della fotocamera corrente, apre l'obiettivo e prepara il dispositivo per lo scatto. In questo momento possono essere impostati flash ed altre opzioni;

`CB_ExitReleaseControl(cc)`: esce dalla **modalità scatto**, riportando infine la fotocamera in uno stato di stand-by con l'obiettivo chiuso;

`CB_Release(cc, Filename)`: scatta la fotografia e la salva nel path specificato dalla stringa `Filename`.

Vi sono inoltre funzioni accessorie per ottenere informazioni o impostare opzioni. Di seguito veranno presentate le due funzioni per gestire il flash:

`[struct Mds, struct Cmps] = CB_GetFlashSettings(cc)`: la funzione restituisce due liste:

- la struttura `Mds` contenente i valori ammissibili del flash;
- la struttura `Cmps` contenente i possibili valori dicompensazione dell'esposizione.

I valori ammissibili per le due liste sono contraddistinti da una stringa descrittiva e da un codice numerico. Il tipo struct vale `cell(string FlashString, integer FlashCode)` per `Mds`, `cell(string CmpString, integer CmpCode)` per `Cmps`.

`CB_SetFlash(cc, Mds, MdI, Cmps, CmpI)`: imposta i valori di flash e compensazione basandosi sugli indici `MdI` e `CmpI` riferiti alle liste in input alla funzione `Mds` e `Cmps`.

Da notare che queste due funzioni possono essere usate solamente quando la fotocamera è in **modalità scatto**.

La funzione Matlab che scatta la foto in remoto si chiama, con poca fantasia, `getPhoto`; la sua *signature*[2] è `[filepath]=getPhoto(path,flash)` dove:

- `path` in input è una stringa che identifica il path dove si vuole salvare l'immagine;
- `flash` in input è un booleano che identifica se l'utente desidera o meno avere il flash;

[2]per *signature* si intendono il nome della funzione, gli input e gli output, formattati come segue: `output=nomefunzione(input)`.

- `filepath` in output identifica il path in cui si è effettivamente salvato il file. Se la variabile `filepath` coincide con la varibile `path` significa che l'operazione è andata a buon fine, altrimenti `filepath` vale la stringa "error".

Si può trovare il sorgente completo in appendice a A.1.1.

Qui si illustra il funzionamento del codice tralasciando i dettagli superflui.

Innanzitutto, si deve fornire a Matlab la directory corrente corretta. Ciò avviene interrogando Matlab con `cd()`; questa funzione restituisce la directory corrente in cui Matlab sta lavorando. Dato che Matlab sta lavorando sull'M-file relativo alla funzione getPhoto, il valore restituito da `cd()` è la directory di getPhoto. A questo punto si compone una nuova directory completando il valore ottenuto con `\CamBox130`, directory in cui si trova la libreria *CameraBox*.

Si possono, a questo punto, usare liberamente le funzioni della libreria. In primis si avvia *CameraBox* con il comando `cc = InitCameraBox`. In seguito si ricercano i discpositivi connessi con `lista = CB_GetConnectedCamera Names(cc)`, dove `lista` contiene i riferimenti a tutte le fotocamere connesse. Successivamente si cicla su `lista` tramite l'indice i, e, ad ogni iterazione:

1. si cerca di connettersi alla fotocamera con `CB_OpenSource(cc, i)`;
2. si cerca di attivare la modalità scatto con `CB_EnterReleaseControl (cc)`;
3. si scatta la foto con `CB_Release(cc, path)`, dove `path` è il valore in input alla funzione.

A questo punto se l'operazione è ultimata con successo bisogna terminare il tutto attraverso diversi passaggi:

1. si ritorna alla modalità stand-by con `CB_ExitReleaseControl(cc)`;
2. si disconnette la fotocamera con `CB_CloseSource(cc)`;
3. si chiude cameraBox con `DestroyCameraBox(cc)`.

Può capitare che l'utente imposti dall'interfaccia grafica la presenza del flash. A livello di codice questo viene fatto controllando la variabile in input `flash` quando si è in modalità scatto, ma prima dell'effettivo scatto della fotografia. Se `flash` è settato a `True` si controllano in prima battuta i

valori possibili per il flash e per la compensazione con `[modes,comp] = CB_GetFlashSettings(cc)`; si impostano in seguito due valori predefiniti per il flash e per la compensazione con `CB_SetFlash(cc, modes, 2, comp, 10)`. Gli indici delle liste in input alla funzione `CB_SetFlash` sono impostati a `2` per il flash, che significa, nella fotocamera Canon A60 utilizzata, il flash normale, e a `10` per la compensazione, che identifica il valore +0.

Quanto finora descritto rappresenta, però, una situazione semplificata; si deve, infatti, tenere conto dei possibili errori. Per risolvere il problema bisogna, dopo ogni istruzione, interrogare il sistema con il comando `{[isErr,ErrType]= CB_IsError(cc)`. Se si verifica un errore, cioè se la variabile `isErr` è impostata a `True`, bisogna agire di conseguenza a seconda del punto del codice in cui si è riscontrato l'errore. Se l'errore si verifica prima del ciclo `for` bisogna:

- prima impostare l'uscita a `filepath='error'`
- poi reimpostare con `cd(curr)` la directory corrente a quella iniziale salvata in `curr`
- infine terminare lo script.

Se si riscontra invece all'interno del ciclo, l'errore potrebbe essere provocato dal dispositivo sbagliato, non compatibile con le SDK; in questo caso basterà saltare alla successiva iterazione del ciclo.

Se alla fine del ciclo non è stata scattata alcuna foto, il flag di controllo `done` rimarrà impostato a `False` e ciò bloccherà l'istruzione `filepath = path,` comando che assegna la variabile in input `path` alla variabile in output `filepath` decretando la buona riuscita dell'operazione. Al termine dello script bisogna ricordarsi di reimpostare la directory corrente di Matlab al valore iniziale.

Si può infine terminare lo script e passare a valutare lo script Matlab di elaborazione della curva.

5.2 Script Matlab: funzione getCurve

Il modo migliore per capire una funzione è conoscerne la *signature*. Pertanto questa sezione inizia introducendo la funzione `getCurve`:

```
[vert]=getCurve(path, number, point, alpha, beta,
   gamma, tang, binorm)
```

dove gli input sono:

`path`: è la stringa che definisce il path dell'immagine da cui ricostruire la curva;

`number`: è un intero che identifica da quanti punti debba essere costituita la curva;

`point`: è un vettore tridimensionale che identifica il punto nello spazio x,y,z da cui si vuole far partire la curva.

`alpha`: è un *float*[3] che rappresenta l'angolo di partenza della curva sul piano xy espresso in gradi;

`beta`: è un *float* che identifica l'angolo iniziale della tangente della curva riferito all'asse xz; come `alpha` anche `beta` è espresso in gradi

`gamma`: è un *float* che individua l'angolo iniziale della normale alla curva riferito all'asse yz, sempre espresso in gradi;

`tang`: è un vettore tridimensionale che identifica l'asse x del sistema di riferimento in cui poggia la curva;

`binorm`: è un'altro vettore tridimensionale che identifica l'asse z del sistema di riferimento in cui poggia la curva.

mentre l'output è:

`vert`: è una matrice $9 \times n$ dove n è il numero dei punti; ogni riga è composta da tre coordinate per la tangente sx, tre per il punto, tre per la tangente dx.

Una volta definiti gli input e gli output della funzione si passa a esaminarne il contenuto interno. Nel seguito verranno descritte le parti sviluppate nell'ambito di questa tesi, tralasciando l'algoritmo di ricostruzione che viene fornito come un dato di fatto, per maggiori informazioni sullo stesso si veda [25] in bibliografia. Inoltre, l'algoritmo di ricostruzione sul quale si è lavorato è in via di ottimizzazione da parte del suo autore, sarebbe quindi inutile descriverne in questa sede una versione obsoleta. Anche in appendice, si veda A.1.2, le parti non sviluppate per questa tesi saranno omesse.

[3] float è un tipo numerico predefinito in quasi tutti i linguaggi di programmazione, identifica un numero con la virgola

5.2.1 Aspetti generali della programmazione Matlab

Innanzitutto bisogna introdurre alcuni aspetti tipici della programmazione in ambiente Matlab. Una funzione si differenzia da uno script eseguibile per la parola chiave `function` all'inizio dello script. Vi sono parecchie differenze concettuali tra script e funzioni.

Lo script permette la visibilità esterna delle sue variabili interne, basta conoscere il nome di una variabile per accedervi dopo l'esecuzione dello script; in questo lo script si comporta in tutto e per tutto come un blocco di codice da importare all'occorrenza.

La funzione, invece, lascia trasparire all'esterno solo il valore dell'output.

Lo script non ha la possibilità di avere valori in input, ma ha visibilità delle variabili dell'ambiente principale di Matlab; in via teorica è possibile passare argomenti a uno script accordandosi sul nome delle variabili. Può capitare, però, che lo script sovrascriva una variabile utile nell'ambiente Matlab, falsando tutti i risultati successivi.

Nelle funzioni, invece, gli unici input sono gli argomenti passati alla stessa; non ci sono quindi rischi di contaminare il codice successivo.

Per queste ragioni si è deciso di trasformare lo script eseguibile iniziale, questa era, infatti la forma dello script Matlab su cui si doveva lavorare, in una funzione.

Una caratteristica importante in Matlab è la possibilità di lasciare alla `function` argomenti opzionali. Questo viene gestito attraverso una variabile speciale chiamata `nargin`. Questa variabile identifica il numero dei parametri che vengono passati in input alla funzione; facendo un controllo su questo valore è possibile permettere alla funzione di ottenere comportamenti diversi in base al numero di parametri in input.

Nel caso di `getCurve` la possibilità di gestire argomenti opzionali è stata sfruttata per differenziare due casi: nel primo, con solo due parametri in ingresso, `path` e `number`, si vuole che la funzione restituisca la curva nel suo sistema di riferimento iniziale, senza l'applicazione di alcuna rototraslazione; nel secondo, in caso cioè vi siano tutti gli otto parametri, si vuole l'applicazione di tutte le trasformazioni specificate dai parametri in ingresso. Materialmente questo comportamento si ottiene controllando la variabile `nargin` e impostando un flag `modcoord` al valore 0, se gli argomenti sono due, al valore 1, se ci sono tutti gli input possibili.

5.2.2 Prima parte: dall'immagine alla Spline

Da questo punto dello script inizia la parte di gestione dell'algoritmo di ricostruzione che, per prima cosa, apre il file immagine identificato da `path`, ridimensionandola ad un valore di default per snellire l'algoritmo. Si estraggono quindi i contorni con l'algoritmo di Canny-Deriche, e si ricostruiscono i punti tridimensionali della curva che vangono salvati nella matrice $n \times 3$ chiamata `centers`, dove n è il numero di punti campionati sulla curva.

La funzione Matlab usata per creare la spline è `[pp,p] = csaps(x,y)` dove `x` è il vettore dell' ascissa, `y` il vettore dell'ordinata, `pp` la spline nella `ppform` e `p` lo *smoothing factor*[4]. Suddetta funzione opera creando un'interpolazione spline cubica definita dai punti (x, y) con uno *smoothing factor* di default. Nel caso tridimensionale `x` identifica l'ascissa curvilinea, mentre `y` rappresenta i punti 3d della curva. La funzione `csaps` matematicamente ricerca la funzione f tale da minimizzare la funzione:

$$p \sum_{j=1}^{n} |y(:, j) - f(x(j))|^2 + (1 - p) \int \left|D^2 f(t)\right|^2 \, dt \tag{5.1}$$

dove p è lo *smoothing factor* calcolato di default sulla base dell'ascissa in ingresso, dove $|z|^2$ identifica la somma dei quadrati di tutti i componenti di z, e dove n è la dimensione di x. Infine, $D^2 f(t)$ indica la derivata seconda di $f(t)$.

Si può comprendere ora il ruolo dello *smoothing factor*, nell' equazione 5.1 rappresenta un peso sul calcolo della spline:

- se `p` è vicino a 1, al limite 1, significa che l'equazione da minimizzare sarà $\sum_{j=1}^{n} |y(:, j) - f(x(j))|^2$, quindi la spline risultante si occuperà di passare necessariamente per i punti definiti da y noncurandosi del rumore presente sui dati;
- se `p` è vicino a 0, al limite 0, la funzione da minimizzare è $\int \left|D^2 f(t)\right|^2 \, dt$ che significa che la spline tenderà ad annullare la derivata seconda, appiattendo il rumore e rendendo la spline più morbida, al limite una retta.

Lo *smoothing factor* rappresenta quindi un fattore da prendere in alta considerazione, considerato il fatto che i dati elaborati dall'algoritmo di ricostruzione non sono esenti da rumore e da errori. Bilanciare p diventa l'unico

[4] lo *smoothing factor* identifica l'approssimazione che viene eseguita per rendere la curva più dolce, ha un valore compreso tra 0 e 1

modo per poter filtrare i risultati grezzi dell'algoritmo di ricostruzione, si sceglierà quindi un valore di p adatto a preservare le caratteristiche della curva e a eliminare il più possibile il rumore.

É ora necessario definire cosa si intende per spline nella `ppform`. Una spline nella `ppform` p definita dalla funzione Matlab:

```
f(t)=polyval(coefs(j,:),t-breaks(j))
```

dove breaks$(j) \leq t <$ breaks$(j+1)$, dove i *breaks* sono gli intervalli in cui la spline è divisa e dove i coefs sono i coefficenti delle polinomiali in ogni intervallo definito dai breaks. La funzione Matlab `polyval(a,x)`, invece, identifica l'equazione matematica:

$$\sum_{j=1}^{k} a(j)x^{k-j} = a(1)x^{k-1} + a(2)x^{k-2} + \cdots + a(k-1)x^1 + a(k) \qquad (5.2)$$

nel caso trattato in questa sede k vale 4 essendo la spline definita da cubiche.

Il valore `pp` restituito da `csaps` sarà una spline nella forma `ppform`, strutturalmente composta da tanti breaks quanti sono i punti del vettore delle ascisse `x` e da tanti coefs quanti sono i breaks moltiplicato per 12; per ogni dimensione vi devono essere 4 coefficienti di definizione della cubica.

Identificata la struttura della funzione che si userà, bisogna trovare i valori di input corretti da assegnare alla funzione. Se `y` è definito dalla matrice `centers`, `x` deve essere l'ascissa curvilinea sulla curva definita da `centers`. Per ottenere l'ascissa curvilinea, prima si calcola il differenziale di `centers` tramite l'istruzione Matlab `df=diff(centers)`, poi si calcola `t`, il vettore rappresentante l'ascissa curvilinea, calcolando la distanza tra i vari punti e sommandola progressivamente tramite l'istruzione `t=cumsum([0,sqrt([1 1 1]*(df.*df)')])`. Matematicamente le istruzioni precedenti svolgono:

$$t(k) = \sum_{i=1}^{k-1} \sqrt{(c_x(i+1) - c_x(i))^2 + (c_y(i+1) - c_y(i))^2 + (c_z(i+1) - c_z(i))^2} \qquad (5.3)$$

dove c_x, c_y, c_z rappresentano rispettivamente le tre colonne di `centers`, dove k varia tra 2 e il numero di punti di `centers`, $2 \leq k \leq$`size(centers,1)`, e dove $t(1) = 0$.

Ottenuta in questo modo l'ascissa curvilinea, si chiama l'espressione `[spine, r] = csaps(t, centers')` per ottenere il valore di default dello *smoothing factor*, `r`; mentre `spine` non serve a nulla. A questo punto si chiama una variante della funzione `csaps` definita come segue: `newy = csaps(x,y,sf,newx)`, dove `newy` è la matrice dei punti che la funzione calcolata da `csaps(x,y)` restituisce per l'ascissa definita dal vettore `newx`,

`sf` è lo *smoothing factor*. Usando le variabili fin qui calcolate si ha `newy = csaps(t,centers',r/splinesmooth,newx)` dove `newx = linspace(t(1), t(end), number)`, mentre `splinesmooth` è una costante predefinita il cui valore è 1500. Questa operazione viene effettuata per far si che la spline sia composta da esattamente tanti breaks quanti sono quelli desiderati con il valore `number` in input; infatti, i breaks della spline si tramutano automaticamente in punti della curva Bézier. Dunque con `linspace(t(1), t(end), number)` si ottiene una diversa composizione dell'ascissa `t` che inizia e finisce con lo stesso valore, ma contiene esattamente `number` valori.

Non resta altro ora che chiamare nuovamente `spline = csaps(newx, newy, 1)` per ottenere la spline definitiva composta dal numero di punti voluto; si è assegnato il valore 1 per lo *smoothing factor* poichè newy, essendo calcolato sulla base di una spline già ammorbidita, è stato già filtrato dal rumore, e non si vuole filtrarlo ulteriormente.

5.2.3 Seconda parte: dalla spline alla Bézier

Nella sezione 4.2.2 si è trovato un sistema di quattro equazioni che permette la trasformazione da spline a Bézier, sistema 4.8, non resta che applicarlo in questa sezione utilizzando la sintassi Matlab.

Per prima cosa si crea la matrice `vert` vuota, ma di dimensioni corrette: `vert = zeros(size(spline.breaks, 2), 9)`; `vert` sarà quindi una matrice $n \times 9$ con n il numero dei breaks della spline [5].

In secondo luogo si estraggono i coefficenti dalla spline, `coef = spline.coefs`, che formeranno una matrice $3 \cdot n \times 4$ composta come segue:

$$\text{coef} = \begin{bmatrix} a_{x\text{breaks}(1)} & b_{x\text{breaks}(1)} & c_{x\text{breaks}(1)} & d_{x\text{breaks}(1)} \\ a_{y\text{breaks}(1)} & b_{y\text{breaks}(1)} & c_{y\text{breaks}(1)} & d_{y\text{breaks}(1)} \\ a_{z\text{breaks}(1)} & b_{z\text{breaks}(1)} & c_{z\text{breaks}(1)} & d_{z\text{breaks}(1)} \\ a_{x\text{breaks}(2)} & b_{x\text{breaks}(2)} & c_{x\text{breaks}(2)} & d_{x\text{breaks}(2)} \\ \vdots & \vdots & \vdots & \vdots \\ a_{z\text{breaks}(n)} & b_{z\text{breaks}(n)} & c_{z\text{breaks}(n)} & d_{z\text{breaks}(n)} \end{bmatrix} \tag{5.4}$$

con n il numero di breaks nella spline.

Si applica a questo punto il sistema 4.8. Si crea un ciclo `for` che scorra tutta la matrice `vert` fino alla penultima riga. A ogni iterazione si calcolano:

la tangente destra del punto corrispondente alla riga iesima (colonne dalla 7 alla 9), nel sistema $P_1 = \frac{1}{3}c\Delta\text{break}_i + d$, in Matlab `vert(i,7:9)= coef(3*(i-1)+ (1:3),3)*d/3 + coef(3*(i-1)+ (1:3),4);`

[5] il valore 2 nell'istruzione `size(spline.breaks, 2)` indica che si vuole la dimensione sulle colonne del vettore $1 \times n$ `spline.breaks`

la tangente sinistra del punto successivo alla riga iesima (colonne dalla 1 alla 3), nel sistema $P_2 = \frac{1}{3}b\Delta\text{break}_i^2 + \frac{2}{3}c\Delta\text{break}_i + d$, in Matlab `vert(i+1,1:3)= coef(3*(i-1)+ (1:3),2)*d^2/3 + coef(3*(i-1)+ (1:3),3)*2*d/3 + coef(3*(i-1)+ (1:3), 4);`

il punto successivo alla riga iesima (colonne dalla 4 alla 6), nel sistema $P_3 = a\Delta\text{break}_i^3 + b\Delta\text{break}_i^2 + c\Delta\text{break}_i + d$, in Matlab `vert(i+1,4:6)= coef(3*(i-1)+ (1:3),1)*d^3 + coef(3*(i-1)+ (1:3),2)*d^2 + coef(3*(i-1)+ (1:3),3)*d + coef(3*(i-1)+ (1:3),4);`

facendo attenzione a due casi particolari:

al primo ciclo di iterazione bisogna calcolare il primo punto (colonne dalla 4 alla 6): nel sistema $P_0 = d$, in Matlab `vert(1,4:6)= coef(1:3, 4);` inoltre, bisogna impostare la tangente sinistra del primo punto equivalente allo punto stesso;

all'ultimo ciclo di iterazione bisogna impostare la tangente destra del punto successivo alla riga iesima equivalente al punto dell'ultima riga della matrice `vert` (riga `i+1`).

Si è riusciti così a ottenere la matrice `vert` corrispondente alla curva nel sistema di riferimento originale. Se nell'interfaccia grafica è impostato il *Toggle* chiamato *Orig*, lo script termina qui è restituisce al chiamante la matrice `vert`; altrimenti...si vedrà nella sottosezione successiva.

5.2.4 Terza parte: dal s.d.r. originale al s.d.r. definito dall'utente

Per prima cosa si controlla il valore della variabile booleana `modcoord`. Se il valore è 1, cioè `True`, bisogna proseguire modificando il sistema di riferimento.

Come operazione preliminare si correggono i valori dei tre angoli in input:

prima si fa il modulo degli angoli rispetto a 360, si fa quindi `alpha = mod( angle, 360)`, in modo da ottenere angoli che assumono valori da 0 a 360 gradi.

poi si portano gli angoli in radianti, `angle = angle*2*pi /360`, ricordando la ben nota equivalenza $x_{\text{gradi}} : 360 = y_{\text{radianti}} : 2\pi$.

A questo punto si centra il primo punto della curva nell'origine sottraendo a tutti i punti della matrice `vert` il primo punto della curva. Questo comportamento si ottiene:

- prima salvando separatamente il primo punto in una nuova variabile, `newpoint = [vert(1,4)vert(1,5)vert(1,6)]`;
- poi ciclando su tutte le righe della matrice, su tutti i punti 3d rappresentati e su ogni coordinata spaziale, si sottrae ill punto `newpoint`. L'istruzione Matlab sarà `vert(i,3*k+j)= vert(i,3*k+j)+ newpoint(j)` dove:
  - `i` rappresenta le righe;
  - `3*k`, con $0 \leq k \leq 2$, rappresenta i 3 punti bezier;
  - `j` la coordinata in esame, x, y e z.

Si calcolano, successivamente, le derivate prima e seconda della spline e il loro valore nel primo punto della curva. Per farlo si utilizzano le funzioni Matlab:

1. `fder = fnder(f)`: dove `fder` è la funzione derivata della funzione `f`;
2. `y = fnval(f,x)`: dove `y` è il valore che la funzione `f` assume in `x`.

Quindi si applica due volte la funzione `fnder` e si valutano le due funzioni ottenute nel primo punto della curva. In Matlab si scrive in questo modo:

- `splined = fnder(spline)`: dove `splined` è la funzione derivata della funzione `spline`;
- `splined2 = fnder(splined)`: dove `splined2` è la seconda derivata della funzione `spline`.
- `funcder = fnval(splined,spline.breaks(1)+eps`: dove `funcder` è il valore della funzione derivata `splined` calcolato nel primo punto della curva, rappresentato dal primo break della spline. Viene aggiunta la quantità infinitesimale `eps` per correggere l'anomalia derivante dalla non continuità sinistra del primo punto della curva.
- `funcder2 = fnval(splined2,spline.breaks(1)+eps`: dove l'output, `funcder2`, è il valore della funzione derivata seconda `splined2` calcolato nel primo punto della curva; questo viene corretto di un fattore ϵ tendente a 0 per ottenere la derivabilità seconda del punto della curva.

Si può ora calcolare la *TNB frame* nel primo punto della curva, come descritto nella sezione a pagina 51. Questo si fa con tre semplici passaggi:

1. `tangente = funcder/norm(funcder)`: si ottiene il vettore tangente normalizzando la funzione derivata;

2. `binormale = cross(funcder, funcder2)/norm(cross(funcder, funcder2))`: si ottiene il vettore binormale normalizzando il prodotto vettoriale[6] tra derivata e derivata seconda;

3. `normale = cross(binormale,tangente)`: si ottiene il vettore normale attraverso il prodotto vettoriale tra binormale e tangente.

Si costruisce la rototraslazione totale componendo le tre rototraslazioni descritte in 4.2.3:

1. `trans = [[tang;0] [cross(binorm,tang);0] [binorm;0] [point;1]]`: si imposta il sistema di riferimento iniziale a quello definito dai valori di input `tang`, `binorm` e `point`; da notare che i valori `tang` e `binorm` devono essere normalizzati.

2. `trans = trans*[cos(alpha)-sin(alpha)0 0; sin(alpha)cos(alpha) 0 0; 0 0 1 0; 0 0 0 1]*inv([cos(beta)0 sin(beta)0; 0 1 0 0; -sin(beta)0 cos(beta)0;0 0 0 1])*[1 0 0 0; 0 cos(gamma)-sin(gamma)0; 0 sin(gamma)cos(gamma)0;0 0 0 1]`: si applicano le tre rototraslazioni definite da α, β e γ.

3. `trans = trans*inv([[tangente;0] [normale;0] [binormale;0] [0; 0; 0; 1]])`: si annulla il sistema di riferimento implicito nella curva definito dalla *TNB frame* nel primo punto della spline.

Per ultima cosa si applica la trasformazione ottenuta ai punti della curva contenuti in `vert`, avendo l'accortezza di trasformare tutti i punti in coordinate omogenee; per fare ciò basta aggiungere il valore 1 nel vettore contenente le coordinate del punto. In Matlab:

- `vertsx = trans*[vert(:,1:3)'; ones(1,size(vert,1))]`
- `vertctr = trans*[vert(:,4:6)'; ones(1,size(vert,1))]`
- `vertdx = trans*[vert(:,7:9)'; ones(1,size(vert,1))]`

Si compone infine l'output come segue:

- `vert = [vertsx(1:3,:)' vertctr(1:3,:)' vertdx(1:3,:)']`

Si conclude così lo script Matlab restituendo in output la matrice `vert`.

[6] `cross(x,y)` equivale a $x \times y$

5.2.5 Precedenti tentativi in getCurve

In questa sezione verranno trattati i tentativi fatti prima di arrivare alla versione definitiva dello script.

Si tratta di idee sperimentali non sempre pienamente corrette; sono state però utili a formulare la versione definitiva fin qui discussa.

Passaggio indiretto da spline a Bézier

Come primo tentativo di passaggio da spline a Beziér si è cercato di lavorare sulla funzione matematica definita dalla spline più che sulla struttura della stessa.

In primo luogo si calcolavano i valori delle funzioni spline, derivata prima della spline e derivata seconda della spline, nei punti definiti dalla partizione a n elementi dell'intervallo che va dal primo all'ultimo breaks della curva; dove n indicava il numero,in input alla funzione, di punti della curva. Si ottenevano in questo modo tre vettori con n elementi, chiamati F, F' e F''.

A questo punto era immediato ottenere i punti della curva, esattamente uguali a F. Il problema principale era ora ottenere i punti che identificavano le tangenti destre e sinistre dei punti della curva. Per calcolarli era necessario introdurre un'ulteriore concetto: la curvatura κ. Il concetto di curvatura, descritto in [26], viene definito da due equazioni:

$$\frac{dT}{ds} = \kappa N \tag{5.5}$$

$$\kappa(t) = \frac{\|x''(t) \times x'(t)\|}{\|x'(t)\|^3} \tag{5.6}$$

L'equazione 5.5 definisce $\kappa(t)$ uno scalare indicatore della variazione della tangente nel tempo; l'equazione 5.6 fornisce un modo per calcolarla.

Il concetto di curvatura era necessario per pesare correttamente la tangente, infatti nelle curve Bézier le tangenti destre e sinistre non hanno modulo unitario, bensì una lunghezza proporzionale alla distanza tra il punto in esame e il punto vicino e inversamente proporzionale alla curvatura della curva; maggiore è la curvatura nel punto iesimo, minore deve essere la lunghezza della tangente.

Definito ciò, si era calcolato i punti delle tangenti destre e sinistre in questo modo:

$$T_{\mathrm{sx}}(t) = F(t) - \frac{F'(t) \cdot \|F(t) - F(t-1)\|}{2 \cdot \|F'(t)\| \cdot (\kappa(t) \cdot 10 + 1)} \tag{5.7}$$

$$T_{\mathrm{dx}}(t) = F(t) + \frac{F'(t) \cdot \|F(t+1) - F(t)\|}{2 \cdot \|F'(t)\| \cdot (\kappa(t) \cdot 10 + 1)} \tag{5.8}$$

dove $2 \leq t \leq n$ per 5.7 e $1 \leq t \leq n-1$ per 5.8 con n uguale al numero di punti della curva; da notare che tutte le variabili, eccetto $\kappa(t)$ hanno tre componenti spaziali, x, y e z. Vi sono tre fattori correttivi che necessitano di spiegazione:

il 2 **nel denominatore** : andava collegato alla distanza dal punto successivo o precedente, $\|F(t) - F(t-1)\|/2$, in quanto si voleva che la lunghezza iniziale della tangente fosse metà della distanza tra i punti confinanti.

il 10 **nel denominatore** : fattore moltiplicativo calcolato sperimentalmente, serviva a pesare la curvatura $\kappa(t)$ in modo che si possa visualizzarne gli effetti sulla lunghezza della tangente; si è osservato come il metodo di calcolo funzionasse a patto di aggiungere un peso sulla curvatura, visualizzando una stessa curva con 300 punti o 30 punti il metodo otteneva risultati eccellenti sovrapponendo esattamente le due curve.

il +1 **al denominatore** : questo fattore additivo sulla curvatura serviva per evitare l'esplosione della lunghezza della tangente; questo fattore permetteva di imporre un limite sulla lunghezza della tangente, ad esempio, con il fattore correttivo la tangente destra nei flessi, dove la curvatura vale 0, è lunga la metà della distanza con il punto successivo, senza tende all'infinito.

In questo modo la lunghezza della tangente destra varia da $0 < \|T_{\mathrm{dx}}(t) - F(t)\| \leq \|F(t+1) - F(t)\|/2$, specularmente la lunghezza della tangente sinistra $0 < \|F(t) - T_{\mathrm{sx}}(t)\| \leq \|F(t) - F(t-1)\|/2$.

Calcolate così le tangenti e i punti bastava comporre la matrice da restituire ed il processo era finito.

Si è scelto di cambiare metodo per non sovrapporre errori derivanti da un calcolo indiretto della curva Bézier. Con il metodo definitivo la curva calcolata è una, definita dalla spline, mentre la curva Bézier viene ottenuta trasformando la spline. Con il metodo vecchio, invece, veniva calcolata prima la spline, poi sulla base dell'analisi della funzione rappresentata dalla stessa venivano calcolati i punti della curva Bézier.

Calcoli trigonometrici per la rotazione della curva

In una fase iniziale del progetto veniva attutata un'unica trasformazione alle coordinate dei punti: a ogni punto veniva sottratta la media delle diverse coordinate x, y e z. In questo modo la curva risultava centrata nell'origine degli assi, punto $(0, 0, 0)$.

Per gestire la costruzione di strutture a più curve, si era pensato di gestire la curva da inserire attraverso due angoli, α e β, che indicassero la direzione della tangente iniziale. Questa operazione veniva gestita da una funzione separata da `getCurve`:

```
[vertices]=modcoord(vert,point,tang,alpha,beta)
```

dove:

`vert`: era la matrice $n \times 9$ rappresentante la curva Bézier da modificare, con i punti di tangenti sx e tangenti dx;

`point`: rappresentava il punto sulla curva originale dove si voleva aggiungere un'altra curva;

`tang`: rappresentava il punto della tangente dx del punto, `point`, sulla curva originale;

`alpha`: l'angolo α sul piano xy che si voleva separasse le due curve;

`beta`: l'angolo β sul piano xz che si voleva separasse le due curve;

`vertices`: la curva Bézier modificata.

La funzione `modcoord` chiamata subito dopo aver creato, con `getCurve`, la curva da aggiungere, usava per i suoi calcoli formule trigonometriche. Prima si centrava il primo punto della curva nell'origine, poi si calcolava la tangente della curva originale normalizzando la differenza tra `tang` e `point`.

A questo punto il vettore tang era formato da tre coordinate normalizzate; se si normalizzavano le proiezioni delle coordinate x e y sul piano xy rispetto alle sole coordinate x e y si ottenevano i valori del $\cos \alpha_{\text{tang}}$ la nuova x e $\sin \alpha_{\text{tang}}$ la nuova y, dove α_{tang} era l'angolo della tangente sul piano xy. Specularmente si ottenevano con lo stesso procedimento su x e z il $\cos \beta_{\text{tang}}$ e il $\sin \beta_{\text{tang}}$, dove βtang era l'angolo della tangente sul piano xz.

Ricordando le formule trigonometriche seguenti:

$$\cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta \tag{5.9}$$
$$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta \tag{5.10}$$
$$\cos(\alpha-\beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta \tag{5.11}$$
$$\sin(\alpha-\beta) = \sin\alpha\cos\beta - \cos\alpha\sin\beta \tag{5.12}$$

era possibile calcolare l'angolo derivante dalla somma della tangente in input con gli angoli α e β:

$$\cos(\alpha_2) = \cos\alpha\frac{T_x}{\sqrt{T_x^2+T_y^2}} - \sin\alpha\frac{T_y}{\sqrt{T_x^2+T_y^2}} \tag{5.13}$$
$$\sin(\alpha_2) = \sin\alpha\frac{T_x}{\sqrt{T_x^2+T_y^2}} + \cos\alpha\frac{T_y}{\sqrt{T_x^2+T_y^2}} \tag{5.14}$$
$$\cos(\beta_2) = \cos\beta\frac{T_x}{\sqrt{T_x^2+T_z^2}} - \sin\beta\frac{T_z}{\sqrt{T_x^2+T_z^2}} \tag{5.15}$$
$$\sin(\beta_2) = \sin\beta\frac{T_x}{\sqrt{T_x^2+T_z^2}} + \cos\beta\frac{T_z}{\sqrt{T_x^2+T_z^2}} \tag{5.16}$$

Giunti a questo punto si doveva trovare l'angolo da sommare alla tangente iniziale della curva per fare in modo che la tangente iniziale venisse definita dagli angoli α_2 e β_2. Per trovare questo angolo bisognava prima calcolare la tangente iniziale della curva sottraendo le coordinate del primo punto alla tangente destra, successivamente normalizzandola prima rispetto ai piani x e y per ottenere il cos e il sin dell'angolo della tangente rispetto al piano xy, poi rispetto a x e z ottenendo il cos e il sin dell'angolo della tangente rispetto al piano xz.

Ottenuti cos e sin degli angoli della tangente iniziale della curva, si trovava cos e sin dell'angolo differenza tra la `tang` in input, trasformata con gli angoli α e β, e la tangente iniziale in questo modo:

$$\cos(\alpha_3) = \cos\alpha_2\frac{\mathrm{Tin}_x}{\sqrt{\mathrm{Tin}_x^2+\mathrm{Tin}_y^2}} + \sin\alpha_2\frac{\mathrm{Tin}_y}{\sqrt{\mathrm{Tin}_x^2+\mathrm{Tin}_y^2}} \tag{5.17}$$
$$\sin(\alpha_3) = \sin\alpha_2\frac{\mathrm{Tin}_x}{\sqrt{\mathrm{Tin}_x^2+\mathrm{Tin}_y^2}} - \cos\alpha_2\frac{\mathrm{Tin}_y}{\sqrt{\mathrm{Tin}_x^2+\mathrm{Tin}_y^2}} \tag{5.18}$$
$$\cos(\beta_3) = \cos\beta_2\frac{\mathrm{Tin}_x}{\sqrt{\mathrm{Tin}_x^2+\mathrm{Tin}_z^2}} + \sin\beta_2\frac{\mathrm{Tin}_z}{\sqrt{\mathrm{Tin}_x^2+\mathrm{Tin}_z^2}} \tag{5.19}$$
$$\sin(\beta_3) = \sin\beta_2\frac{\mathrm{Tin}_x}{\sqrt{\mathrm{Tin}_x^2+\mathrm{Tin}_z^2}} - \cos\beta_2\frac{\mathrm{Tin}_z}{\sqrt{\mathrm{Tin}_x^2+\mathrm{Tin}_z^2}} \tag{5.20}$$

dove Tin è la tangente iniziale della curva da aggiungere.

Trovato a questo punto cos e sin degli angoli da sommare bisognava applicare la trasformazione a ogni punto della curva. Per fare questo:

1. si calcolava il vettore con origine nel primo punto della curva che puntasse al punto in esame, chiamato P
2. si normalizzava questo vettore
   (a) prima rispetto alle coordinate x e y ottenendo cos e sin dell'angolo del vettore sull'asse xy
   (b) poi rispetto alle coordinate x z ottenendo cos e sin dell'angolo del vettore sull'asse xz
3. si sommava a questo punto gli angoli ottenuti, usando le equazioni trigonometriche 5.9 e 5.10, con gli angoli α_3 e β_3 ottenendo il vettore ruotato prima sul piano xy e poi sul piano xz
4. a questo punto si ritornava alle coordinate x, y e z rispettivamente:
   (a) la coordinata x moltiplicando il cos dell'angolo formato dal vettore ruotato sul piano xy per la norma sulle coordinate x e y del vettore P
   (b) la y moltiplicando il sin dell'angolo formato dal vettore ruotato sul piano xy per la norma sulle coordinate x e y del vettore P
   (c) la z moltiplicando il sin dell'angolo formato dal vettore ruotato sul piano xz per la norma sulle coordinate x e z del vettore P
5. si sostituiva al punto in esame e si passava al successivo, prendendo anche i punti delle tangenti dx e sx.

Matematicamente, semplificando:

$$x = (\cos\alpha_3(P_x - O_x) - \sin\alpha_3(P_y - O_y)) + O_x \tag{5.21}$$

$$y = (\sin\alpha_3(P_x - O_x) + \cos\alpha_3(P_y - O_y)) + O_y \tag{5.22}$$

$$z = (\sin\beta_3(P_x - O_x) + \cos\beta_3 P_z - O_z) + O_z \tag{5.23}$$

Si otteneva così una nuova matrice di punti ruotati secondo gli angoli α e β rispetto alla tangente della curva originale. Comportamento eguale a quello voluto.

Questo metodo di calcolo della rotazione presentava però due gravi difetti:

- non si teneva conto in alcun modo della normale alla curva, ciò voleva dire non poter controllare dove finisse l'asse y dopo le trasformazioni;

- il sistema risultava impreciso: il risultato era preciso solo rispetto a uno dei due piani xy o xz contenendo un errore sull'altro piano. La correttezza dell'uno rispetto all'altro dipendeva da quale equazione veniva utilizzata per calcolare l'asse x, se $x = (\cos\alpha_3(P_x - O_x) - \sin\alpha_3(P_y - O_y)) + O_x$, il risultato era preciso rispetto all'asse xy, se $x = (\cos\beta_3(P_x - O_x) - \sin\beta_3(P_y - O_y)) + O_x$ il risultato era preciso rispetto all'asse xz. Questo errore era dovuto alla non corretta integrazione delle due rotazioni, viste erroneamente come separate, e non gestite in maniera coordinata.

Il metodo rappresenta l'idea iniziale che ha introdotto il concetto di rototraslazione della curva. Risulta quindi un punto d'inizio importante per lo sviluppo effettivo del software.

Per queste ragioni e per purezza concettuale si è passati al sistema di rototraslazione matriciale.

5.3 Chiamata delle funzioni Matlab da Python

Nello scorso capitolo si è visto come la comunicazione tra Python e Matlab venga gestita da una libreria Python esterna chiamata MatlabWrap.

Nella presente sezione verranno descritte prima le modalità d'uso della libreria, poi l'effettivo impiego nel software realizzato per la tesi.

5.3.1 Matlabwrap: modalità d'uso

Nella sezione 4.3 si è descritto il funzionamento della libreria Matlabwrap, qui di seguito si discuterà di come effettivamente Matlabwrap si usa.

Matlabwrap è molto semplice da usare. Una volta importata la funzione mlab dal modulo mlabwrap, a inizio script, viene automaticamente aperta la schermata di Matlab connessa alla libreria; in Python l'istruzione si scrive `from mlabwrap import mlab`. A questo punto basta, da qualsiasi parte del programma Python, scrivere l'istruzione Matlab, voluta preceduta da `mlab.`, per ottenere il comportamento desiderato, ad esempio `mlab.abs(y)` per ottenere il valore assoluto di y.

Per la gestione degli input e output da Matlab, mlabwrap fa uso di una

libreria esterna di Python chiamata `numpy`. Per rendere funzionale mlabwrap bisogna, quindi, importare anche suddetta libreria attraverso il comando `import numpy`. Questo modulo permette la gestione di strutture numeriche complesse, come matrici e array; queste, infatti, le funzionalità usate da mlabwrap. Alla chiamata di una semplice funzione Matlab, ad esempio `z = mlab.cross(x,y)` per il prodotto vettoriale, gli input, x e y, devono essere del tipo array definito nel modulo numpy, e il valore restituito dalla funzione Matlab, z, sarà dello stesso tipo.

Il tipo array lavora come una lista Python contenente solo float, può essere bidimensionale per gestire le matrici e può essere usata in un ciclo `for` per scandirne gli elementi. Per creare un oggetto array basta scrivere `numpy.array([3 2],[2 3])` per creare una matrice bidimensionale; se si importa il tipo array in una maniera differente, attraverso il comando `from numpy import array` si può creare un array semplicemente digitando `array([1 0 0],[0 1 0],[0 0 1])` per creare la matrice identità.

Da notare come se si richiama la funzione Python `help` su un comando Matlab, ad esempio `help(mlab.norm)`, viene restituito all'utente Python l'help di Matlab per la funzione ricercata.

5.3.2 Matlabwrap: uso in matCurveFunct

Il modulo mlabwrap viene importato in uno solo dei moduli Python scritti per il presente lavoro: in **matCurveFunct.py**; per il codice completo vedi A.2.1.

Dal lato Python la comunicazione con Matlab viene gestita tutta nel modulo **matCurveFunct.py**.

Si inizia importando i gli oggetti array e mlab da numpy e mlabwrap, `from numpy import array` e `from mlabwrap import mlab`; operazione che, come si è spiegato, inizializza Matlab e lo connette al programma Python. Da notare che se Matlab è gia stato aperto da una precedente chiamata di mlabwrap, non è necessario riavviarlo.

Si usa successivamente l'oggetto mlab nelle funzioni vectFromMatlab e getPhoto.

Chiamata di getCurve(...) da vectFromMatlab(...)

La funzione vectFromMatlab ha la seguente signature:

```
vect = vectFromMatlab(fpath, points=20, point=None,
   alpha=None, beta=None, gamma=None, tang=None,
   binorm=None)
```

dove i parametri sono:

fpath : identifica il path dell'immagine da passare alla funzione getCurve di Matlab per essere elaborata;

points : intero opzionale, con valore di default pari a 20, che identifica il numero di punti che si vuole abbia la curva, anche questo da passare direttamente a getCurve;

point : tupla opzionale con tre elementi float, identifica il punto dove si vuole abbia origine la curva, si vedrà in seguito come viene convertito per essere passato a getCurve;

alpha : angolo della tangente iniziale della curva sul piano xy, identificato da un float opzionale, può essere passato direttamente alla funzione Matlab getCurve;

beta : angolo della tangente iniziale della curva sul piano xz, anch'esso identificato da un float opzionale, come alpha può essere passato direttamente a getCurve;

gamma : angolo della normale iniziale della curva sul piano yz, viene identificato, come alpha e beta, da un float opzionale, come gli altri due angoli è possibile passarlo a getCurve direttamente;

tang : tupla opzionale di tre float, identifica l'asse x del sistema di riferimento della curva, va modificato per poter essere passato a getCurve;

binorm : tupla opzionale di tre float, identifica l'asse z del sistema di riferimento della curva, come tang va modificato per poter essere passato a getCurve.

restituisce, invece:

vect : una matrice $n \times 9$ contenente gli n punti della curva con rispettive tangenti, viene rappresentata da una lista bidimensionale.

Liste vs Tuple A questo punto è opportuno chiarire la differenza in Python tra tipo lista e tipo tupla, fonte [29]:

- **tipo lista**: in Python rappresenta una lista dinamica modificabile contenente qualsiasi tipo di dato, anche mischiato. Viene istanziata tramite parentesi quadre: `lista1=[12, 32, 2, 'pinco', 32.4, 'pallino']`. Di seguito alcune proprietà delle liste:
  - Possono essere scorse attraverso un ciclo `for` con l'istruzione `for i in lista1`, che restituisce in `i` tutti gli elementi uno per uno.
  - Possono essere aggiunti elementi tramite l'istruzione `lista1.append('prova')`; dopo l'istruzione `lista1` sarà `[12, 32, 2, 'pinco', 32.4, 'pallino', 'prova']`.
  - Possono essere selezionati solo pezzi di lista con l'istruzione `lista[a:b]` che selezionerà gli elementi da a compreso a b non compreso, l'ordine degli elementi parte da 0; ad esempio:
    - l'istruzione `lista1[2:4]` restituirà `[2,'pinco']`;
    - l'itruzione `lista1[:3]` restituirà i primi tre elementi `[12,32,2]`;
    - mentre l'istruzione `lista1[3]` restituirà il quarto elemento di `lista1` cioè `'pinco'`.
  - É possibile, inoltre creare liste multidimensionali inserendo liste come elementi di una lista; data ad esempio `lista2=['alpha',23]`, tramite l'istruzione `lista1.append(lista2)` si otterrà `lista1` eguale a `[12, 32, 2, 'pinco', 32.4, 'pallino', 'prova', ['alpha', 23]]`.
  - Si può modificare e cancellare il contenuto di ogni cella della lista; ad esempio
    - l'istruzione `lista1[4:]=[]` cancellerà tutti gli elementi dal quarto in poi, la lista risultante sarà `[12,32,2,'pinco']`;
    - l'istruzione `lista1[:3]=[69]` restituirà la lista `[69, 'pinco']`
  - Infine, si può conoscerne la lunghezza tramite l'istruzione `len(lista1)` che per la lista `lista1` restituirà il valore 2.
- **tipo tupla**: le tuple, a differenza delle liste, rappresentano un tipo di dato immutabile, non è perciò possibile aggiungere o eliminare elementi dopo l'istanziazione. Si istanziano tra parentesi tonde e possono contenere qualsiasi tipo di dato: `tup1=(23,'hello',32.23)`. Queste alcune proprietà:

- Si possono selezionare gli elementi come si faceva con le liste, tramite l'istruzione `tupla[a:b]`;
- Possono essere multidimensionali; è possibile creare un'altra tupa in questo modo: `tup2=((1,2,3),tup1,4)` che renderà `tup2` eguale a `((1,2,3),(23,'hello',32.23),4)`.
- Si può conoscere la lunghezza di una tupla tramite l'istruzione `len(tupla)`;
- Si può ciclare in una tupla come si fa con una lista, attraverso un ciclo `for`.

Come si è visto, i tipi lista e tupla sono molto simili tra loro, ma presentano una sostanziale, quanto importante differenza, il secondo è immutabile, il primo no; ciò rende le liste più versatili, ma meno sicure delle tuple, che si è sicuri non possano mai venire modificate.

Si è scelto in questa sede di usare le tuple per purezza concettuale, infatti le tuple vengono di solito usate per costruire insiemi numerici di coordinate, ad esempio `point(2.8,34.2,12.3)` che identifica un punto nello spazio 3d; in queste occsioni vengono preferite tuple a liste proprio per la loro immutabilità. Quando si ha a che fare con coordinate spaziali non si vuole correre il rischio che vengano modificate all'interno del programma, le tuple permettono di avere una certa sicurezza.

Funzionamento vectFromMatlab Si passa ora a esaminare il funzionamento interno della funzione vectFromMatlab.

Passo iniziale è cambiare la *current directory* di Matlab, infatti, una volta aperto da mlabwrap, Matlab ha la cd impostata alla directory corrente dal quale è stato lanciato lo script. Prima bisogna verificare se la cd di Matlab è già stata modificata nel modo corretto, ciò si fa verificando la sempice condizione `mlab.cd()!= SCRIPTPATH + r"\Matlab"` dove SCRIPTPATH è una variabile globale preimpostata alla directory dov'è situato MatlabCurve. Se il controllo fallisce si procede reimpostando la cd al valore `SCRIPTPATH + r"\Matlab"` directory in cui sono salvati gli script Matlab.

Prima di fare ciò è necessario disattivare un parametro del modulo mlab, attraverso l'istruzione `mlab._autosync_dirs = False`. Il flag `mlab._autosync_dirs` predispone in mlab il blocco della cd, attuato da mlabwrap, a detta dei programmatori, per evitare spiacevoli inconvenienti. In questa sede è però necessario cambiare cd a Matlab, in quanto le funzionalità richieste derivano dall'utilizzo di M-files esterni alle normali librerie di Matlab.

Si può ora, dopo aver disattivato il flag, impostare la nuova cd tramite l'istruzione `mlab.cd(SCRIPTPATH + r"\Matlab")`.

É possibile a questo punto richiamare la funzione Matlab getCurve. Per farlo bisogna però accertarsi della presenza o meno dei parametri opzionali, e, in base a questo, chiamare la funzione getCurve in maniera differente.

Se vi sono solo i parametri `fpath` e `points` diversi da `None`: si chiama la versione semplice di getCurve tramite l'istruzione `numarr = mlab.getCurve(fpath, points)` dove numarr sarà un array bidimensionale contenente i punti della curva nella sua forma originale, non rototraslata.

Se vi sono in più i parametri `alpha`, `beta` e `gamma` diversi da `None`: la funzione viene richiamata così `numarr = mlab.getCurve(fpath, points, array((0, 0, 0)), alpha, beta, gamma, array((1, 0, 0)), array((0, 0, 1)))` dove per default si è impostato il valore del punto di origine all'origine degli assi e gli assi tang e binorm corrispondenti agli assi x e z naturali;

Se vi è anche `point` diverso da `None`: viene incluso il parametro point nella funzione getCurve ottenendo `numarr = mlab.getCurve(fpath, points, array(point), alpha, beta, gamma, array((1, 0, 0)), array((0, 0, 1)))`; come si vede point non può essere passato a getCurve normalmente, ma va trasformato tramite il costruttore di array.

Se, infine, tutti gli argomenti sono diversi da `None`: la funzione getCurve viene passata con tutti i parametri `numarr = mlab.getCurve(fpath, points, array(point), alpha, beta, gamma, array(tang), array(binorm))`

Si riesce così ad ottenere l'array numarr contenente la curva in forma Bézier.

Ora non rimane altro da fare che trasformarla in forma di lista di liste per terminare la funzione. Inizialmente si crea una lista vuota `vect=[]`. In seguito si popola di elementi quest'ultima scorrendo una per una le righe e di li i punti di numarr e, parallelamente, inserendoli nel corretto ordine in vert. Dato che il tipo array ammette il ciclo `for` risulta immediato scorrere tutte le righe di numarr, creare delle liste temporanee k, scorrere tutti gli elementi delle righe e inserirli nella liste k, per poi inserire le liste k come elementi nella lista vect.

Ottenuta la lista vect, si restituisce quest'ultima in output e si termina la funzione.

Chiamata da Python di getPhoto(...) in Matlab

La signature di getPhoto in Python è molto simile a quella in Matlab:

```
path = getPhoto(direct,flash)
```

dove i parametri in input sono:

direct : a differenza di getPhoto in Matlab dove il primo parametro rappresenta il path del file che si vuole creare, completo di nome del file, qui direct identifica la directory della cartella in cui si vuole salvare l'immagine, sarà nella forma `"c:\blabla\etciu\qui\"`; importante il carattere \ a fine stringa.

flash : come getPhoto in Matlab rappresenta un Booleano che identifica l'opzione flash.

mentre l'output è:

path : la path finale, comprensiva del nome del file, dove si è salvata l'immagine scattata.

Per prima cosa, come in vectFromMatlab, bisogna verificare se la cd è corretta, e in caso negativo correggerla.

Si verifica, a questo punto, se la variabile in input è la stringa vuota, in caso affermativo si imposta la directory a una directory di default predefinita: la cartella photo nella cartella dove è contenuto lo script qui in esame, definita da `SCRIPTPATH + "\photo"` dove SCRIPTPATH è la variabile globale contenente la directory dello script.

Successivamente:

1. si apre la cartella definita da direct tramite la funzione di sistema `dct=os.listdir(direct)` che restituisce in dct la lista dei file contenuti in direct;

2. si predispone un nome file con l'istruzione `fi="image-%(num)05d.jpg"` composto da una parte fissa definita da "image-....jpg"e da un contatore a cinque cifre.

3. si cicla, a questo punto, sul contatore nel nome del file dell'immagine attraverso un ciclo `while` con la condizione `fi%{"num":i} in dct`; che identifica la presenza in dct di un file con il nome riferito al contatore attuale. All'interno del ciclo, semplicemente, si incrementa il contatore.

Usciti dal ciclo, il valore del contatore definisce un nome file valido non presente nella directory, che composto con direct, forma un filepath valido e pronto ad essere utilizzato in getPhoto di Matlab.

Solo a questo punto è possibile richiamare la funzione getPhoto tramite l'istruzione `newpath=mlab.getPhoto(direct+fi%{"num":i},flash)`.

Per terminare la funzione rimane soltanto da restituire in output il valore restituito da getPhoto di Matlab, cioè newpath, che ricordiamo varrà il filepath in cui si è slavata la fotografia se lo scatto è avvenuto correttamente, la stringa "error"altrimenti.

5.3.3 Precedenti tentativi di comunicazione Python-Matlab

Prima di adottare mlabwrap sono stati sperimentati altri metodi di comunicazione tra i due processi definiti da Matlab e Python.

Comunicazione tramite file

L'unico tentativo di successo si serviva di file esterni per la comunicazione Matlab-Python. Si vedrà nei paragrafi seguenti come funzionava il metodo dai punti di vista prima di Python e poi di Matlab:

- In **Python**: la comunicazione veniva gestita in una funzione apposita chiamata `runMatlab` che aveva in input un solo parametro: `fpath` che indicava il path dell'immagine da elaborere.
  Come prima cosa si apriva in scrittuta un file nella directory dello script chiamato flag, vi si scriveva il valore 0 e si chiudeva il file; operazioni effettuate rispettivamente dai comandi `fi=open(SCRIPTPATH + "/flag", 'w')`, `fi.write('0')` e `fi.close()`.
  Il file flag veniva utilizzato come un semaforo, finchè il valore scritto al suo interno era 0, significava che Matlab non aveva ancora finito di creare la curva, quando Matlab lo impostava a 1 significava che Matlab aveva finito di elaborare e aveva salvato la curva in un'altro file.
  In secondo luogo si lanciava Matlab con la chiamata di sistema `os.system("Matlab␣ -sd␣"+SCRIPTPATH + "/Matlab"+'␣-r␣"main('+"'"+fpath+"'"+')"␣ -nodesktop␣-nosplash')`; questa chiamata funzionava come una riga da shell di comando di windows, chiama Matlab con parametri:

  - `-sd path/Matlab`: identificava la cd di Matlab all'avvio, veniva definita dalla path dove erano situati gli script;
  - `-r main('fpath')`: chiamava la funzione Matlab main, successivamente chiamata getCurve, con parametro fpath all'avvio di Matlab;
  - `-nodesktop`: impostava la schermata di Matlab alla sola shell di comando;

  - `-nosplash`: evitava che all'avvio di Matlab venisse mostrata la schermata di intorduzione.

  A questo punto si controllava il file di flag finchè il valore letto non fosse risultato 1. Questa operazione si articolava in questi passaggi:

  - si apriva il file di flag in lettura tramite il comando `fi = open(SCRIPTPATH + "/flag", 'r')`;
  - si iterava con un ciclo `while` sulla condizione `float(fi.read())!= 1.0`
  - all'interno del ciclo si facevano attendeva 10 secondi e si ritornava a inizio file, rispettivamente attraverso le istruzioni `time.sleep(10)` e `fi.seek(0)`;
  - se si usciva dal ciclo significava che era stato letto 1 e si terminava la funzione.

  Una volta terminata la funzione runMatlab era possibile leggere il file vert, contenente la curva in formato Bézier elaborata da Matlab, con l'assoluta certezza che Matlab aveva già scritto il suo output.

- In **Matlab**: Dopo essere stato lanciato attraverso l'istuzione sopra descritta Matlab compiva le sue elaborazioni. Una volta costruita la curva, veniva salvato il risultato nel file vert strutturato secondo una matrice di float $n \times 9$. Questa operazione veniva effettuata attraverso l'istruzione `save -ascii ../vert vertices` dove vertices è la variabile contenente la matrice.
  In seguito si salvava la variabile flag, impostata a 1, nel file chiamato flag. Ciò avveniva usando l'istruzione `save -ascii ../flag flag`. Si terminava a questo punto lo script Matlab.

In questo modo si riusciva con successo a coordinare i due processi.

Il metodo però non risultava pulito concettualmente, avvalendosi di una struttura esterna del filesystem per funzionare, si è quindi optato per una libreria che mascherasse la comunicazione Matlab-Python all'utente.

Comunicazione diretta

Si erano provate altre strade per la gestione dei processi. Una di queste, la più auspicabile, prevedeva l'utilizzo della funzione di sistema operativo `spawnl`.

Si pensava di permettere la comunicazione tra i due processi attraverso la gestione dei processi del sistema operativo: si voleva chiamare

Matlab attraverso le funzione `pid = os.spawnl(P_NOWAIT,..\Matlab)` e `os.waitpid(pid)`, ciò doveva in teoria permettere di aspettare la terminazione di Matlab prima di proseguire nel programma Python.

Peccato però che Matlab viene avviato in una maniera particolare: viene avviato un processo padre, chiamato Matlab.bat, che genera un figlio, il vero e proprio programma Matlab e termina. Questa particolarità nell'avvio di Matlab rendeva impossibile conoscere il pid del processo figlio, Matlab stesso, e, quindi, rendeva impossibile il metodo di comunicazione sopra presentato.

5.4 Trattamento della curva in Blender

Per poter disegnare la curva in Blender è necessario utilizzare le librerie Python che il software di modellazione 3d offre.

In questa sezione si mostra la struttura delle API di Blender e il loro utilizzo all'interno del software.

5.4.1 Moduli Curve e BezTriple

Il modulo fondamentale per la modellazione delle curve in Blender è chiamato Curve. Si trova tutta la documentazione relativa in [30]. Qui di seguito si descrivono esclusivamente le funzioni in questa sede utilizzate.

Il modulo Curve contiene tre classi di oggetti al suo interno: **Curve**, **CurNurb** e **SurfNurb**. Nel presente lavoro sono state utilizzate solo le prime due:

Curve : rappresenta un oggetto contenitore di più curve, permette di impostare parametri globali su tutte le curve inserite nell'insieme;

CurNurb : rappresenta la singola curva costituita da una sequenza di punti.

All'interno del modulo sono presenti solo due funzioni:

- `New(name)`: funzione che crea un nuovo oggetto **Curve** nominato secondo la stringa `name`;
- `Get(name=None)`: funzione che restituisce in output l'oggetto **Curve** con nome `name`, se il parametro `name` è stato passato alla stessa, una lista con tutte le curve presenti nella scena corrente altrimenti.

Una volta creato l'oggetto **Curve** tramite la funzione `Curve.New('cur')` è possibile modificare l'oggetto a piacimento secondo una serie di funzioni:

- `appendNurb(newpoint)`: funzione fondamentale, aggiunge una nuova curva con punto iniziale `newpoint` all'oggetto **Curve**; `newpoint` è un punto nel formato **BezTriple**, mentre l'output della funzione è il nuovo oggetto **CurNurb** creato e associato a **Curve**;
- `setExt2(ext2)`: imposta il valore di BevelDepth della curva, `ext2` è un float; Bevel Depth indica la dimensione del raggio della struttura tridimensionale cotruita attorno alla curva;
- `setBevresol(bevresol)`: imposta il valore di Bevel Resolution della curva, `ext2` è un float; Bevel Resolution indica il numero di segmenti da cui deve essere formata la sezione della struttura attorno alla curva, maggiore è, più la sezione è circolare. Il numero di segmenti n viene calcolato nel seguente modo $n = 4 + 2 \cdot \text{bevresol}$
- `setFlag(val)`: imposta il valore dei toggle corrispondenti alla curva dipendentemente dai bit componenti l'intero val:
  - bit 0: il toggle "3d"è impostato; identifica se la curva è tridimensionale o meno;
  - bit 1: il toggle "Front"è impostato; se impostato da corpo solo a una metà della struttura 3d attorno alla curva;
  - bit 2: il toggle "Back"è impostato; se impostato permette la consistenza solo all'altra metà della struttura 3d attorno alla curva;
  - bit 3: il toggle "CurvePath"è impostato; predispone la curva a essere un cammino per un'oggetto in un'animazione 3d;
  - bit 4: il toggle "CurveFollow"è impostato; permette agli oggetti associati al cammino della curva di ruotare seguendo la curva stessa;

La classe **CurNurb**, creata con l'istruzione `objc.appendNurb(point)` dove `objc` è un'oggetto **Curve**, permette le funzioni:

- `append(newpoint)`: aggiunge il punto `newpoint`, in formato BezTriple, alla curva contenuta nell'oggetto **CurNurb** che richiama la funzione;
- `__getitem__(n)`: restituisce in output il punto n-esimo della curva in forma **BezTriple**.

Gli oggetti **Curve** permettono l'uso di indici; è quindi possibile selezionare l'i-esima **CurNurb** usando l'istruzione `curve[i]`. Risulta inoltre possibile scorrere tutte le **CurNurb** in un oggetto **Curve** tramite un ciclo `for`.

Si rende necessario ora descrivere il tipo **BezTriple**. Il modulo **BezTriple** permette la gestione dei punti caratterizzanti una curva Bézier, descrive la classe **BezTriple** e ne permette la costruzione attraverso l'istruzione:

- `New(coords)`: restituisce l'oggetto **BezTriple** associato alle 9 coordinate, tre per ogni punto, presenti nella lista `coords`.

Ogni oggetto **BezTriple** contiene al suo interno due attributi utili:

- `vec`: contiene una lista di tre elementi `[handle, knot, handle]` composti anch'essi dalle tre coordinate `[x, y, z]` espresse tramite float;
- `handleTypes`: contiene un valore di default per i vincoli sulle maniglie, nome delle tangenti nelle curve in Blender, definito tra i seguenti:
  - `FREE`: le maniglie non hanno restrizioni, sono indipendenti l'una dall'altra;
  - `AUTO`: le maniglie sono calcolate automaticamente a partire dalla posizione dei nodi;
  - `VECT`: le maniglie puntano verso il punto successivo, quella destra, e verso il punto precedente, quella sinistra;
  - `ALIGN`: costringe le maniglie a giacere sulla stessa retta.

Tramite tutti questi strumenti è possibile avere il controllo totale delle curve nell'ambiente Blender.

Utilizzo in matCurveFunct.py

Oltre alle funzioni `vectFromMatlab` e `getPhoto`, nel modulo **matCurveFunc** vi sono altre funzioni:

- `bezFromVect(vect)`: trasforma la lista definita in `vect`, composta di 9 float, in un'oggetto **BezTriple** restituito in output. Al suo interno semplicemente:
  1. richiama il costruttore di **BezTriple** su `vect`, istruzione `k = BezTriple.New(vect)`
  2. imposta le maniglie al valore FREE, istruzione `k.handleTypes = (BezTriple.HandleTypes.FREE, BezTriple.HandleTypes.FREE)`

3. ritorna il risultato.

- `newCurve(vect)`: crea una nuova curva a partire dalla lista di liste `vect`, contenente le coordinate di tutti i punti della curva Bézier, e la restituisce in output. Lavora attraverso questi passaggi:

  1. per prima cosa si crea un'oggetto **Curve** vuoto, istruzione `cu = Curve.New()`
  2. successivamente si aggiunge un'oggetto **CurNurb** definito dal primo punto di `vect` all'oggetto **Curve** precedentemente creato, istruzione `cu.appendNurb(bezFromVect(vect[0]))`
  3. si seleziona l'oggetto **CurNurb** creato dall'oggetto **Curve**, istruzione `cun=cu[0]`
  4. si cicla sulla lista in ingresso `vect`, attraverso un ciclo `for`, e a ogni iterazione aggiunge il punto i-esimo, istruzione `cun.append(bezFromVect(i))`
  5. si restituisce in output l'oggetto **Curve**.

- `addCurve(cur,vect)`: aggiunge un nuovo oggetto **CurNurb**, definito dalla lista `vect`, a un'oggetto **Curve** già esistente, parametro `cur`. Dopo aver controllato la validità degli input si effettuano le seguenti operazioni:

  1. si aggiunge un nuovo oggetto **CurNurb** vuoto a `cur`, istruzione `cur.appendNurb(bezFromVect(vect[0]))`
  2. si seleziona l'ultima curva aggiunta a `cur`, istruzione `cun = cur[len(cur)-1]`
  3. si cicla sulla lista in ingresso `vect`, attraverso un ciclo `for`, e a ogni iterazione aggiunge il punto i-esimo, istruzione `cun.append(bezFromVect(i))`
  4. si restituisce in output l'oggetto modificato `cur`

- `setTube(cur)`: imposta i flag per rendere la curva un tubo 3d all'oggetto `cur` di tipo **Curve**. Lavora impostando tre diversi valori per tre attributi della curva:

  1. si imposta il valore di Bevel Depth a 1, istruzione `cur.setExt2(1)`
  2. si imposta il valore di Bevel Resolution a 5 pari a 14 segmenti, istruzione `cur.setBevresol(5)`. Questo parametro influisce molto sulla complessità del rendering della scena. Per questo motivo viene pre-impostato a un valore relativamente basso, in modo

da ottenere un'effetto soddisfacente che non aumenti eccessivamente il tempo necessario per effettuare il rendering. Questo, come tutti gli altri valori qui pre-impostati, può essere re-impostato successivamente dalla consolle di Blender con un valore a piacere.

3. si impostano i toggle della curva in modo da lasciare soltanto il toggle della tridimensionalità attivo, istruzione `cur.setFlag(1)`

Termina così la discussione dei contenuti del modulo matCurveFunct.

5.4.2 Modulo MatCurve.py di Matlab Curve

Il modulo **MatCurve**, in appendice a A.2.2, è stato creato per semplificare l'uso delle funzioni presenti in **matCurveFunct.py** costruendo una classe, chiamato **matCurve**, che facesse da interfaccia con le suddette funzioni.

La classe **matCurve** contiene al suo interno il seguente attributo:

- `_curve`: contiene al suo interno l'oggetto **Curve** che contraddistingue l'oggetto **matCurve**.

e i seguenti metodi:

- `__init__(self, path, points=20, point=None, alpha=None, beta=None, gamma=None, tang=None, binorm=None)`: costruttore della classe **matCurve** si limita a costruire un nuovo oggetto assegnando all'attributo `_curve` la curva ricostruita da Matlab; utilizza un'unica istruzione:

  1. `self._curve = newCurve(vectFromMatlab(path, points, point, alpha, beta, gamma, tang, binorm))`: ricostruisce la curva a partire dal vettore restituito dalla funzione `getCurve` di Matlab.

- `makeTube(self)`: semplicemente richiama la funzione `setTube` di **matCurveFunct** sull'unico attributo della classe; istruzione `self._curve = setTube(self._curve)`.

- `getCurve(self)`: restituisce al chiamante la cura contenuta nell'attributo `_curve`.

Da notare che il parametro `self` come primo parametro dei metodi all'interno di una classe identifica l'oggetto stesso e non va inserito al momento della chiamata del metodo.

Qui si conclude la gestione della curva in Blender, nella prossimna sezione si descriverà l'interfaccia che fa uso dei costrutti fin qui esposti.

5.5 Interfaccia grafica di Blender

In questa sezione vengono mostrati i moduli di Blender usati per visualizzare la curva a schermo e per gestire l'interfaccia utente. Come nella precedente sezione vengono mostrate esclusivamente le funzioni effettivamente usate.

In secondo luogo viene mostrato il loro utilizzo all'interno dello script Python contenente il main di Matlab curve, vedi appendice A.2.3.

5.5.1 Modulo Object

Il modulo **Object** definisce la classe **Object** che caratterizza qualsiasi oggetto virtuale di Blender. Sono oggetti Blender le curve, le mesh, le luci, le telecamere...

La classe **Object** definisce alcuni attributi e diversi metodi per la gestione degli oggetti:

- `LocX, LocY, LocZ`: attributi che contengono la posizione dell'oggetto nella scena rispetto agli assi x, y e z;
- `RotX, RotY, RotZ`: attributi che contengono la rotazione dell'oggetto nella scena rispetto agli assi x, y e z;
- `SizeX, SizeY, SizeZ`: attributi che contengono il ridimensionamento moltiplicativo dell'oggetto nella scena rispetto alle dimensioni originali sugli assi x, y e z;
- `getData(nameonly=False, mesh=False)`: restituisce l'oggetto nel formato specifico (**Curve**, Mesh,...); i parametri sono:
  - `nameonly` è un Booleano che, se impostato a `True`, fa si che la funzione restituisca solo il nome dell'oggetto contenuto;
  - `mesh`: Booleano che, se impostato a `True`, retìstituisce la Mesh corrispondente all'oggetto, di qualsiasi tipo sia.
- `setLocation(x,y,z)`: imposta la posizione dell'oggetto secondo i tre valori in input;
- `setSize(x,y,z)`: ridimensiona l'oggetto secondo i tre parametri moltiplicativi in input `x`, `y` e `z`; ad esempio se `x`, `y` e `z` valgono 0.5 la dimensione dell'oggetto viene ridotta a un'ottavo di quella attuale;

Queste sono le funzioni utilizzate nel modulo **main.py** per il controllo degli oggetti nella scena.

5.5.2 Modulo Scene

Il modulo **Scene** permette di accedere alle finestre di visualizzazione 3d di Blender. Nello specifico un'oggetto **Scene** definisce un insieme di oggetti contenuti in una scena. La scena può sia essere quella corrente, cioè quella visualizzata, sia una scena salvata senza essere visualizzata.

Si descrive ora la struttura interna del modulo **Scene**. Come già si è accennato il modulo definisce una classe chiamata **Scene**. Inoltre, il modulo contiene un'ulteriore classe chiamata **SceneObjects** che permette il controllo della lista di oggetti assegnati alla scena. Il modulo contiene poche funzioni usate come costruttori di oggetti **Scene**:

- `New(name='Scene')`: crea una nuova scena chiamata `name` e la restituisce in otuput.
- `Get(name=None)`: ottiene la scena chiamata `name` se il parametro è definito, tutte le scene presenti in Blender altrimenti.
- `GetCurrent()`: restituisce la scena corrente, quella visualizzata nella vista 3d di Blender.

La classe **Scene** fa uso di diversi metodi e attributi, di seguito vengono descritti esclusivamente quelli utilizzati:

- `objects`: attributo che contiene tutti gli oggetti appartenenti alla scena, contiene un oggetto **SceneObjects**
- `link(object)`: aggiunge l'oggetto definito da `object` alla scena.
- `unlink(object)`: cancella l'oggetto definito da `object` alla scena, se l'oggetto viene trovato e cancellato il metodo restituisce `True`, altrimenti `False`.

La classe **SceneObjects**, infine, contiene:

- `active`: attributo che contiene l'oggetto attivo sulla scena, cioè quello correntemente selezionato e modificabile; può essere attivo un solo oggetto alla volta.
- `new(data)`: metodo che consente di aggiungere un nuovo oggetto alla scena; `data` definisce un oggetto nella sua forma originale, può essere **Curve**, **Lamp**, **Mesh**...
- `join(objects)`: associa l'oggetto corrente alla lista di oggetti contenuti in `objects`. Dopo l'istruzione tutti gli oggetti formeranno un'unica

entità. Nota bene che però gli oggetti singoli contenuti in `objects` non vengono cancellati dalla scena, coesisteranno quindi il raggruppamento di oggetti e i singoli oggetti precedenti nella stessa scena.

- `link(object)`: metodo che consente di aggiungere un oggetto già esistente, definito da `object`, alla scena;

- `unlink(object)`: metodo che consente di eliminare un oggetto già esistente, definito da `object`, dalla scena.

Grazie a queste classi e funzioni è possibile manipolare la scena corrente di Blender e aggiungervi qualsiasi tipo di oggetti, nell'ambito del presente progetto curve.

5.5.3 Modulo Window

Il modulo **Window** permette di accedere a una serie di funzioni utili alla gestione delle finestre di Blender.

Nell'ambito del seguente progetto vengono usate solo poche di queste funzioni qui di seguito descritte:

- `FileSelector(callback, title='SELECT␣FILE', filename='<default>')`: la funzione apre la finestra di selezione file, i tre valori di input definiscono:

  - `callback`: indica il nome di una funzione a un solo parametro `f(filename)` identificante un path di un file. La funzione `f(filename)` viene richiamata dal file selector una volta selezionato il file, il cui path viene passato come primo argomento.
  - `title`: stringa che descrive l'operazione da compiere e da nome alla finestra di selezione file, ad esempio "si prega di selezionare un file".
  - `filename`: stringa che identifica un nome di file di default.

- `GetCursorPos()`: restituisce in output una lista contenente la posizione del cursore 3d nella scena, sotto forma di un float per ognuna delle tre coordinate.

- `RedrawAll()`: ridisegna tutte le finestre.

Tramite questo modulo è stato possibile gestire le finestre in input per la selezione dei file nell'interfaccia grafica di Matlab Curve.

5.5.4 Modulo Draw

Il modulo Draw, come già si è visto nella sezione 4.5, permette di gestire l'interfaccia utente in Blender. Esso definisce la classe **Button** e una serie di funzioni per generare tutta la serie di pulsanti definita nel capitolo precedente.

Le funzioni in questa sede utilizzate sono:

- `Exit()`: chiude la finestra disegnata dallo script Python e termina quest'ultimo.
- `Redraw(after=0)`: predispone il sistema a ridisegnare l'interfaccia grafica dopo `after` secondi, dove `after` è un intero.
- `Draw()`: forza il sistema a ridisegnare immediatamente l'interfaccia grafica.
- `Register(draw=None, event=None, button=None)`: registra l'interfaccia grafica definita dalle tre funzioni `draw`, `event` e `button` all'interfaccia di Blender, permettendo allo script di rimanere attivo. Le tre funzioni definiscono:
  - `draw`: definisce la funzione che disegna effettivamente i pulsanti nell'interfaccia grafica, non deve avere input. Al suo interno deve contenere tutte le dichiarazioni dei pulsanti che si vuole siano presenti nell'interfaccia.
  - `event`: definisce una funzione con due valori in input che elenca gli eventi associati ai pulsanti della tastiera o del mouse. I due parametri sono:
    - `evt`: l'identificativo dell'evento che si è verificato; il modulo **Draw** definisce in quantità valori di default tra cui: uno per ogni tasto della tastiera e per ogni combinazione particolare, uno per ogni tasto del mouse,...
    - `val`: intero identifica il rilascio, valore 0, o la pressione, altrimenti, di un tasto o pulsante.
  - `button`: definisce la funzione che permette di gestire gli eventi dei pulsanti, deve avere un singolo parametro chiamato `evt` che definisce l'evento occorso, cioè il pulsante premuto.
- `Create(value)`: crea un oggetto **Button** dipendentemente dal tipo di `value`. Il parametro `value` rappresenta il valore di default e può essere di tipo:

  - `int`: il pulsante creato sarà un **Number** a valori interi, vedi sez 4.5.
  - `float`: il pulsante creato sarà un **Number** a valori decimali.
  - `string`: verrà creata una casella di testo modificabile.

- `PushButton(name, event, x, y, width, height, tooltip=None, callback=None)`: crea un pulsante standard a pressione. I parametri definiscono:
  - `name`: stringa che definisce il nome del pulsante;
  - `event`: intero identificativo dell'evento chiamato dal pulsante;
  - `x`: intero che definisce la posizione orizzontale del pulsante nello schermo calcolata in punti, valore definito partendo dall'angolo in basso a sinistra della finestra riservata in Blender all'interfaccia grafica degli script;
  - `y`: intero che definisce la posizione verticale del pulsante nello schermo calcolata in punti;
  - `width`: intero che definisce la larghezza del pulsante, viene calcolata in punti;
  - `height`: intero che definisce l'altezza del pulsante, calcolata anch'essa in punti
  - `tooltip`: stringa di help che appare quando si passa il mouse sopra il pulsante;
  - `callback`: funzione da richiamare immediatamente dopo la pressione del pulsante, deve accettare i due argomenti `evt` e `val`, di default è impostato a `None` perchè si preferisce gestire gli eventi dei pulsanti tramite la singola funzione definita da `button` in `Register`.

- `PupIntInput(text, default, min, max)`: mostra sovraschermo un selezionatore di intero pop-up[7], un **Number**, che viene chiuso una volta selezionato un valore. I parametri sono:
  - `text`: stringa che identifica il testo mostrato nel pop-up;
  - `default`: intero, identifica il valore di default del pop-up;
  - `min`: intero, identifica il valore minimo selezionabile;

[7] per pop-up si intende una piccola finestra che appare sullo schermo sovrastando tutte le altre finestre

  - `max`: intero, identifica il valore massimo selezionabile.

- `PupBlock(title, sequence)`: mostra sovraschermo un blocco pop-up di pulsanti. I due parametri sono:
  - `title`: stringa, definisce il nome visualizzato a schermo del blocco;
  - `sequence`: lista di elementi che identificano il contenuto del blocco. Possono essere formattati come segue:
    * `string`: definisce una semplice etichetta;
    * `(string, Value, string)`: definisce un toggle in cui Value è un'oggetto **Button**, il primo `string` ne definisce il nome e l'ultimo string, opzionale, definisce il tooltip[8].
    * `(string, Value, min, max, string)`: definisce un **Number** dove string iniziale, Value e string finale valgono come sopra, min, un itero, vale il minimo valore possibile, max, sempre un intero, vale il massimo valore possibile.

- `Toggle(name, event, x, y, width, height, default, tooltip=None, callback=None)`: crea un pulsante **Toggle**, cioè un pulsante con modalità on-off. I parametri valgono:
  - `name`: vedi PushButton(...);
  - `event`: vedi PushButton(...);
  - `x`: vedi PushButton(...);
  - `y`: vedi PushButton(...);
  - `width`: vedi PushButton(...);
  - `height`: vedi PushButton(...);
  - `default`: valore di default del **Toggle**, può valere 1, se lo si vuole impostato a on, 0, se lo si vuole impostato a off;
  - `tooltip`: vedi PushButton(...);
  - `callback`: vedi PushButton(...).

- `Number(name, event, x, y, width, height, initial, min, max, tooltip=None, callback=None)`: crea un pulsante **Number**, cioè un pulsante che permette di scorrere o scrivere direttamente un numeri. I parametri sono:

[8] il tooltip definisce in Blender un'etichetta che appare al passaggio del mouse sul pulsante

  - `name`: vedi PushButton(...);
  - `event`: vedi PushButton(...);
  - `x`: vedi PushButton(...);
  - `y`: vedi PushButton(...);
  - `width`: vedi PushButton(...);
  - `height`: vedi PushButton(...);
  - `initial`: valore iniziale del **Number**, se impostato ad un valore intero il pulsante scorre numeri interi, se impostato a un valore float scorre numeri decimali;
  - `min`: valore minimo del **Number**;
  - `max`: valore massimo del **Number**;
  - `tooltip`: vedi PushButton(...);
  - `callback`: vedi PushButton(...).

- `String(name, event, x, y, width, height, initial, length, tooltip=None, callback=None)`: disegna una casella di testo modificabile. I parametri individuano:
  - `name`: vedi PushButton(...);
  - `event`: vedi PushButton(...);
  - `x`: vedi PushButton(...);
  - `y`: vedi PushButton(...);
  - `width`: vedi PushButton(...);
  - `height`: vedi PushButton(...);
  - `initial`: stringa iniziale contenuta nella casella di testo;
  - `length`: numero massimo di caratteri contenuti nella stringa;
  - `tooltip`: vedi PushButton(...);
  - `callback`: vedi PushButton(...).

La classe **Button** contiene, invece, un singolo attributo:

- `val`: valore contenuto nel pulsante; il tipo di `val` dipende dal tipo di pulsante a cui l'attributo appartiene.

Usando questi strumenti è possibile disegnare in Blender un'interfaccia completa e funzionale.

Si passerà ora a esaminare l'utilizzo dei precedenti moduli nel progetto qui esposto.

5.5.5 Main dello script Python

Il main dello script Python è contenuto nel file **main.py**, il cui sorgente è reperibile in appendice A.2.3.

Nello script **main** è racchiuso sia l'interfaccia grafica che la manipolazione delle curve a video, attuata attraverso i moduli **matCurve** e **matCurveFunct**.

Si analizza qui di seguito il funzionamento di tutto il codice contenuto in **main** soffermandosi sugli aspetti più matematici del processo.

Dopo aver importato dai moduli le classi e le funzioni utili, si passa a introdurre alcune variabili globali, utili per la successiva definizione dei pulsanti. Si creano così le variabili:

value : intero impostato a 20, identifica il numero di default iniziale di punti nella;

alpha, beta, gamma : tre float impostati a 0.0, identificano i valori di default iniziali dei tre angoli α, β e γ di cui tanto si è già discusso.

pT2, pT3, flashT, pToggle : identificano quattro variabili vuote che conterranno i quattro **Toggle**, sotto forma di oggetti **Button**, definiti successivamente nell'interfaccia grafica; i quattro toggle in questione sono quelli trattati nella sezione 4.5 e sono chiamati rispettivamente **Tube**, **Orig**, **Flash** e **Photo**.

tube, orig, flash, camera : valori di default per i quattro **Toggle**, l'unico preimpostato a **True** è `tube`, mentre gli altri sono impostati a `false`.

pathB : variabile vuota che conterrà successivamente la casella di testo modificabile, chiamata **String**, del path in cui si vuole vengano salvate le immagini scattate in remoto.

path : valore di default della casella di testo, inizialmente impostato alla stringa vuota.

Successivamente alla dichiarazione delle variabili globali, si definiscono le funzioni utilizzate:

- `gui()`: questa funzione senza parametri disegna l'interfaccia grafica di Blender.
- `button(evt)`: questa funzione gestisce gli eventi associati ai pulsanti, accetta un unico parametro `evt` che identifica l'evento verificatosi, cioè il pulsante premuto.

- `addcurve(filepath)`: disegna la curva nella scena corrente e costruisce strutture a più curve. Questa funzione viene richiamata alla pressione del pulsante **Curve** subito dopo o aver selezionato il file dal file selector, o aver scattato la fotografia tramite `getPhoto`.

Infine si registrano le funzioni definite nell'interfaccia di blender tramite l'istruzione `Draw.Register(gui,None,button)`.

Qui di seguito vengono descritti i meccanismi interni che permettono alle tre funzioni sopracitate di funzionare.

Funzionamento interno di gui()

La funzione `gui()` si articola in due passaggi:

1. per prima cosa si dichiara l'utilizzo di variabili globali[9]: istruzione `global pToggle, pathB, pT2, flashT, pT3, path, camera, tube, orig, flash.`
2. poi si dichiarano i vari pulsanti dell'interfaccia, per informazioni sul loro utilizzo vedi sezione 4.5:
   (a) il pulsante **Curve**, istruzione `Draw.PushButton("Curve", 1, 10, 100, 200, 20, "draw␣Matlab␣Curve");`
   (b) il pulsante **quit**, istruzione `Draw.PushButton("quit", 2, 10, 25, 200, 20, "exit");`
   (c) il **Toggle Photo**, istruzione `pToggle = Draw.Toggle("Photo", 3, 10, 50, 50, 20, camera, "select␣if␣get␣the␣photo␣from␣camera␣or␣from␣file")`
   (d) il **Toggle Flash**, istruzione `flashT = Draw.Toggle("Flash", 6, 60, 50, 50, 20, flash, "select␣if␣photo␣with␣or␣without␣flash");`
   (e) il **Toggle Tube**, istruzione `pT2 = Draw.Toggle("Tube", 5, 110, 50, 50, 20, tube, "select␣if␣you␣want␣tube␣or␣curve");`
   (f) il **Toggle Orig**, istruzione `pT3 = Draw.Toggle("Orig", 7, 160, 50, 50, 20, orig, "select␣if␣you␣want␣original␣position");`
   (g) la casella di testo **pd**, istruzione `pathB = Draw.String("pd", 4, 10, 75, 200, 20, path, 50, "select␣where␣store␣the␣photo␣if␣none␣store␣in␣scriptpath")`.

[9] in Python, non essendo necessario dichiarare le variabili, bisogna dichiarare espressamente quali variabili si vuole siano globali all'interno di una funzione, altrimenti viene creata una nuova variabile con lo stesso nome e scope diverso

Funzionamento interno di button(evt)

Come la funzione `gui()`, la funzione `button(evt)` si articola in due punti:

1. prima si dichiara l'utilizzo delle variabili globali, esattamente come in `gui()`;
2. successivamente si controlla la variabile `evt`, verificando quindi il pulsante premuto, e in base a questa si agisce in maniera consona:
   (a) se viene premuto **Curve**: prima di tutto si controlla il toggle **Photo**, identificato dalla variabile globale `camera`, e in base a quest'ultima si agisce di conseguenza:
      i. Se `camera` vale 0: si chiama un file selector che a sua volta richiamerà la funzione `addcurve` in **main.py**, istruzione `Window.FileSelector(addcurve,"Selezionare␣immagine:")`;
      ii. Se `camera` vale 1: si chiama prima `getPhoto` di **matCurveFunct**, istruzione `st=getPhoto(path,flash)` dove `path` e `flash` sono le variabili globali corrispondenti, e successivamente, se `st` rappresenta un filepath valido, si chiama la funzione `addcurve`, istruzione `addcurve(st)`.
   (b) se viene premuto **quit**: si chiude l'interfaccia grafica e si termina lo script, istruzione `Draw.Exit()`;
   (c) se viene premuto uno dei quattro **Toggle**: si salva il valore contenuto nel **Toggle** nella variabile globale corrispondente, ad esempio per il toggle **Flash** l'istruzione è `flash=flashT.val`;
   (d) se viene cambiata la stringa nella casella di testo: si tenta di aprire la directory modificata, istruzione `os.listdir(pathB.val)`, se questa operazione ha successo si salva il valore della casella di testo nella variabile corrispondente, istruzione `path = pathB.val`, altrimenti si sovrascrive la casella di testo con il valore precedentemente salvato nella variabile corrispondente, istruzione `pathB.val = path`, e si ridisegna l'interfaccia subito dopo, istruzione `Draw.Redraw(1)`. Qiesto controllo sull' istruzione `os.listdir(pathB.val)` viene fatto attraverso un blocco `try:... except:...`, infatti l'istruzione in questione lancia un'eccezione se il path in input non è valido, mentre continua l'esecuzione se il path è valido.

Funzionamento interno addcurve(filepath)

La funzione `addcurve(filepath)` si articola in diversi passaggi:

1. si dichiara l'utilizzo di alcune variabili globali, istruzione `global value, alpha, beta, gamma, num, point, tube, orig`;

2. si inizializzano alcune variabili locali:

   (a) `block`: lista vuota che conterrà i pulsanti per il blocco pop-up contenente le opzioni della curva;

   (b) `flag`: booleano che identifica la selezione o meno di una curva, inizialmente impostato a `False`;

   (c) `curve`: contenitore vuoto per l'oggetto **matCurve** che verrà creato;

   (d) `obj`: contenitore vuoto per un'oggetto **Object** che verrà creato;

   (e) `num`: intero di default identificativo della curva a cui si vuole associarne una ulteriore, inizialmente impostato a 1 cioè alla prima curva della struttura

   (f) `point`: intero che definisce il punto i-esimo della curva in cui si vuole inserire un'altra curva, inizialmente impostato a 0.

3. si ottiene la scena corrente, istruzione `scn = Scene.GetCurrent()`;

4. si controlla se nello stato corrente di Blender l'oggetto attivo nella scena è una curva, condizione `scn.objects.active!=None and scn.objects.active.getType()=='Curve'`; se lo è si effettua una serie di operazioni:

   (a) si crea un pulsante che definisca la selezione del punto sulla curva, sotto forma di numero intero, istruzione `p = Draw.Create(point)`;

   (b) si imposta il flag chiamato `flag` a `True`, poichè effettivamente vi è una curva selezionata;

   (c) si ottiene l'oggetto attivo nella scena, istruzione `obj = scn.objects.active`;

   (d) si ottiene la curva relativa all'oggetto attivo, istruzione `curve = obj.getData()`

   (e) se la curva non è singola, cioè se l'oggetto **Curve** contiene più di una curva, si disegna un selezionatore di intero pop-up utile per decidere a quale curva, tra quelle nella struttura, debba esserne aggiunta un'ulteriore, istruzione `num=Draw.PupIntInput("selez␣curva:", num, 1, len(curve))`.

5. si creano prima il pulsante definito dalla variabile globale `value`, istruzione `v = Draw.Create(value)`, successivamente, se è selezionato il toggle **Orig**, si creano i pulsanti per `alpha`, `beta` e `gamma`, istruzione, ad esempio, `a = Draw.Create(alpha)`;

6. si riempie la lista `block` con i pulsanti creati:

   (a) per il numero di punti della curva che può variare da 5 a 500, istruzione `block.append(("numero␣di␣punti:", v, 5, 500))`;

   (b) se `flag` vale `True` si appende al blocco il pulsante per la selezione del punto sulla curva in cui si vuole aggiungere un'altra curva, il valore varia dal primo all'ultimo punto della curva, istruzione `block.append(("punto␣in␣cui␣inserire:",p, 0,len(curve[num-1])-1))`;

   (c) se `orig` vale `False` si appendono i pulsanti per la selezione degli angoli α, β e γ, il valore varia da 0 a 360 con una precisione decimale, istruzione, ad esempio per la variabile `alpha`, `block.append(("alpha:", a, 0.0, 360.0))`.

7. una volta preparato la lista `block` si crea il blocco pop-up, istruzione `Draw.PupBlock("Parameters␣of␣the␣curve",block)`;

8. si salvano i parametri selezionati tramite il blocco pop-up nelle variabili globali corrispondenti, operazione che si fa per le variabili `value`, `alpha`, `beta` e `gamma`, istruzione, ad esempio, `value=v.val`;

9. si crea la curva dipendentemente dal boolenao `flag`:

   (a) se `True`: viene aggiunta la curva alla curva selezionata tramite le opzioni precedenti. Per fare questo si svolgono i seguenti passaggi:

      i. si seleziona la curva definita dalla variabile `num` definita dall'utente, tramite il selezionatore di intero pop-up, istruzione `curr = curve[num-1]`;

      ii. si ottiene il punto Bézier, in forma di lista con tre punti, definito dalla variabile `point`, istruzione `ptemp = curr. __getitem__(point). getTriple()`;

      iii. si estraggono le coordinate del punto, istruzione `knot = (ptemp[1][0], ptemp[1][1], ptemp[1][2])`;

      iv. si calcolano il vettore tangente e binormale con il metodo descritto nella sezione 4.4 ottenendo le liste `tang` e `binorm`. Si omette qui il codice in quanto eccessivamente lungo rispetto

agli scopi qui preposti di mostrare l'utilizzo dei moduli di Blender nello script Python. Se si vuole visionare il codice si veda A.2.3.

v. si crea l'oggetto **matCurve**, istruzione `mcur = matCurve(filepath, value, knot, alpha, beta, gamma, tang, binorm)`;

vi. si inserisce la curva nella scena corrente, istruzione `obj2 = scn. objects. new(mcur. getCurve())`;

vii. si trasforma la curva appena creata con le stesse trasformazioni della curva sulla quale deve essere attaccata. Questo serve per correggere l'errore che si presenta se la curva originale è stata modificata con gli strumenti di Blender dopo essere stata creata. Si articola in tre istruzioni:

A. si corregge la posizione, istruzione `obj2.setLocation(obj.LocX, obj.LocY, obj.LocZ)`;

B. si corregge la rotazione, istruzione `obj2.RotX, obj2.RotY, obj2.RotZ = obj.RotX, obj.RotY, obj.RotZ`;

C. si corregge il ridimensionamento, istruzione `obj2.setSize(obj.SizeX, obj.SizeY, obj.SizeZ)`.

viii. si associa la curva appena creata alla curva originale, istruzione `obj.join([obj2])`

ix. si elimina la curva creata nella versione non associata, istruzione `scn.objects.unlink(obj2)`.

(b) se `False`: si crea una nuova curva. Passaggi:

i. si crea l'oggetto **matCurve** in base al toggle **Orig**, se impostato a `True`, l'istruzione è `curve=matCurve(filepath, value)`, altrimenti l'istruzione è `curve = matCurve(filepath, value, None, alpha, beta, gamma)`;

ii. se il toggle **Tube** è selezionato si procede impostando la curva in modo che appaia come un tubo, istruzione `curve.makeTube()`;

iii. si aggiunge la curva alla scena, istruzione `obj = scn.objects. new(curve. getCurve())`;

iv. si ottiene la posizione del cursore 3d, istruzione `pos = Window. GetCursorPos()`;

v. si trasla la curva nella posizione del cursore, istruzione `obj. setLocation (pos[0], pos[1], pos[2])`

Finisce così anche lo script Python per la gestione del Main di Matlab Curve.

5.6 Conclusioni del capitolo

In questo capitolo si è discussa l'implementazione dell'intero processo, si sono visti i vari aspetti di programmazione affrontati e le librerie utilizzate.

Nel prossimo capitolo si discuteranno le prove sperimentali svolte e gli utilizzi che si possono ottenere dal software.

Capitolo 6

Realizzazioni sperimentali e valutazione

DURANTE IL CORSO DEL PRESENTE CAPITOLO vengono descritte le prove sperimentali svolte, comprensive di immagini del materiale prodotto.

Prima di tutto vengono discusse e visualizzate le modalità di rappresentazione della curva nella finestra 3d di Blender, con tutti i possibili utilizzi del software. In questa prima parte vengono, inoltre, mostrate le immagini utilizzate nella fase di testing, accompagnate dai risultati ottenuti.

Secondariamente verranno introdotte le possibilità di utilizzo del software.

Infine, si valuta il lavoro prodotto individuando casi specifici in cui il software fallisce, esaminano inoltre alcune possibilità di miglioramento dello stesso.

6.1 Sperimentazione e visualizzazione 3D

L'immagine più utilizzata nella fase di testing durante lo sviluppo del progetto è la fotografia di un cavo di alimentazione da computer nero su sfondo bianco, immagine 6.1.

Per buona parte delle viusalizzazioni 3D nella presente sezione viene usata l'immagine 6.1 come sorgente.

Inizialmente, la visualizzazione della curva 3D consisteva nella diretta trasposizione della curva visualizzata precedentemente in Matlab. Suddetta curva, costruita disegnando dei cerchietti perperdicolari alla curva in ogni suo punto di definizione, dava forma ad una specie di molla, rappresen-

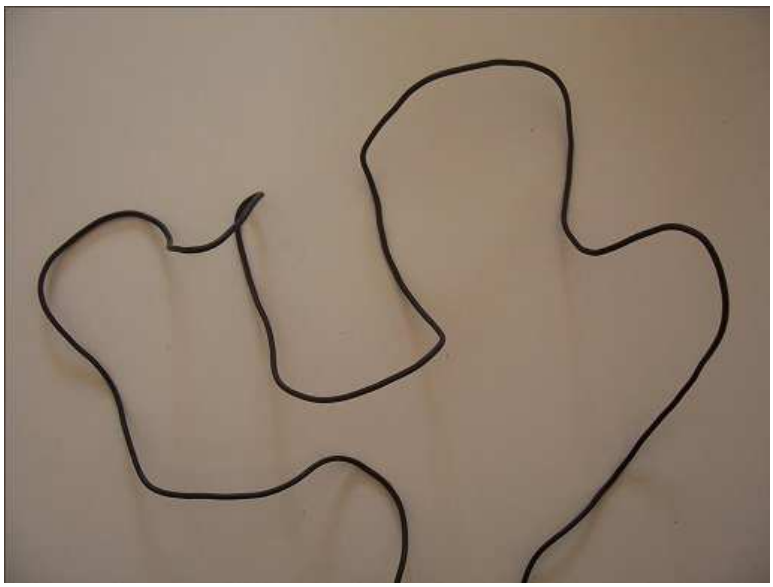
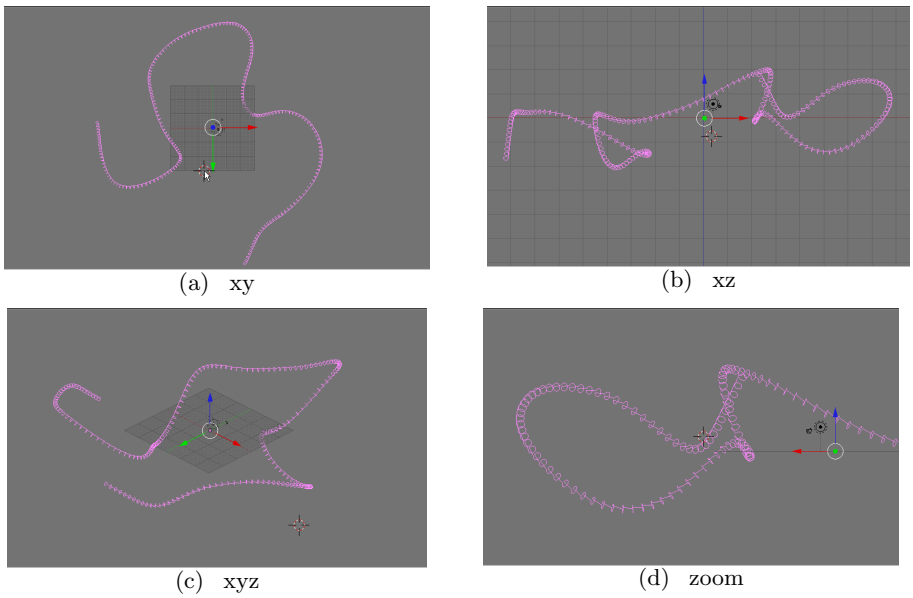


Figura 6.1: immagine per il testing

tante la struttura tridimensionale della curva. I risultati di queste prime approssimazioni sono visualizzate nella figura 6.2.



(a) xy (b) xz

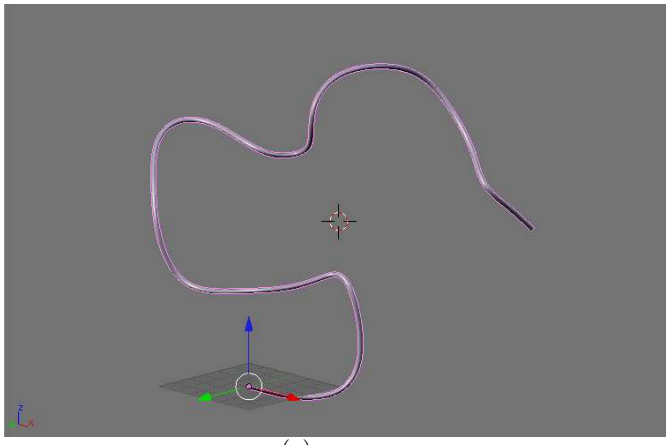
(c) xyz (d) zoom

Figura 6.2: prima modalità di visualizzazione della curva in Blender

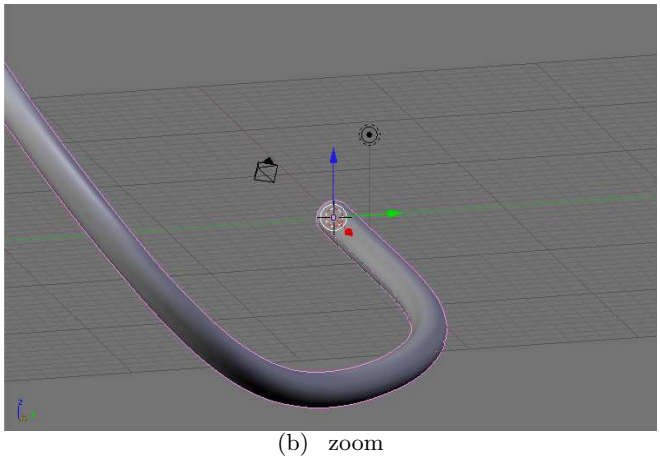
Le immagini nella figura 6.2 rappresentano rispettivamente:

1. la curva ricostruita a partire dall'immagine 6.1 sul piano xy;
2. la curva sul piano xz;
3. la curva nello spazio tridimensionale;
4. un particolare della curva per visualizzare la struttura a cerchietti.

Successivamente la struttura della curva è stata modificata, usando le funzionalità di Blender, in modo da permettere l'utilizzo della curva per il rendering, figure 6.3. Questa nuova struttura consiste nell'estrusione della curva, cioè nella creazione di una superficie attorno alla curva.



(a) curva



(b) zoom

Figura 6.3: seconda modalità di visualizzazione della curva in Blender

La nuova struttura creata è resa opzionale dal toggle **Tube**, pulsante

di opzione booleana nell'interfaccia utente. La curva visualizzata quando il toggle **Tube** è disattivato è una semplice linea 3d, figura 6.4.

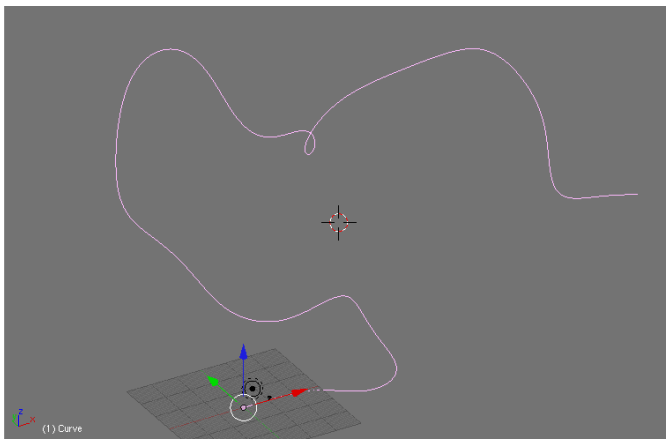


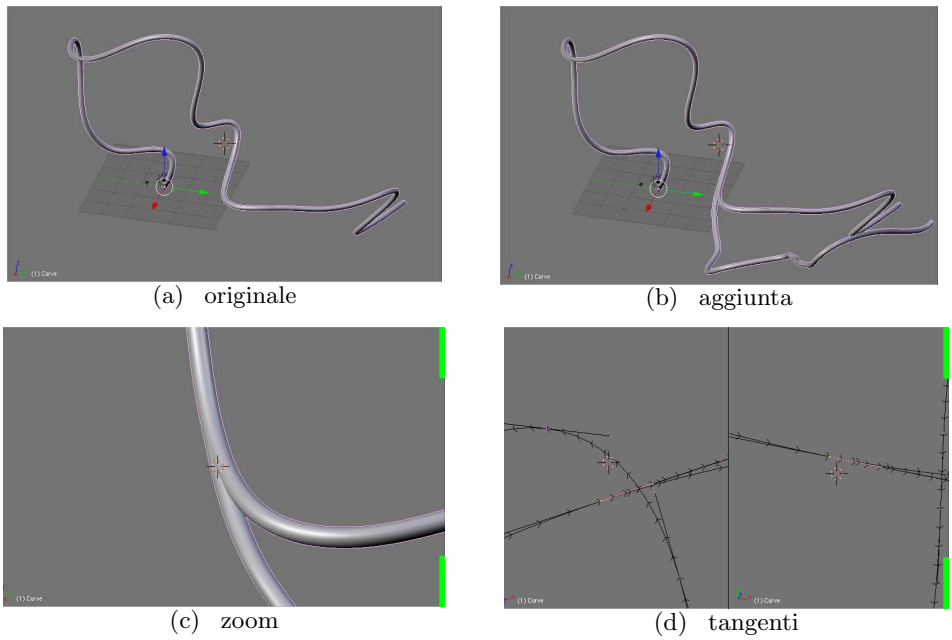
Figura 6.4: curva come linea 3D

L'algoritmo per l'innesto di una curva su di un'altra è stata una fase critica del testing, ma ha permesso di trovare parecchi errori. Nello specifico, grazie al testing è stato possibile accorgersi di rotazioni sbagliate e di piani di riferimento errati. Nella figura 6.5 è possibile visualizzare l'algoritmo in funzione.

Le immagini in figura 6.5 rappresentano un inserimento al trentesimo punto di una curva composta da cinquanta punti con valori di α, β e γ pari a 0 gradi. Le immagini rispettivamente rappresentano:

1. la curva originale
2. la curva composta tramite l'innesto di una seconda curva in mezzo alla prima
3. particolare del punto di innesto
4. le tangenti sovrapposte viste da due posizioni diverse, le tangenti sovrapposte sono la tangente dx del trentesimo punto della prima curva e la tangente dx del primo punto della seconda curva.

In questa fase di sviluppo del software si è iniziato ad usare altre immagini per ricostruire curve tridimensionali, l'immagine 6.6, ad esempio, rappresenta il tubicino bianco con anima in fil di ferro descritto nel capitolo 3.



(a) originale (b) aggiunta

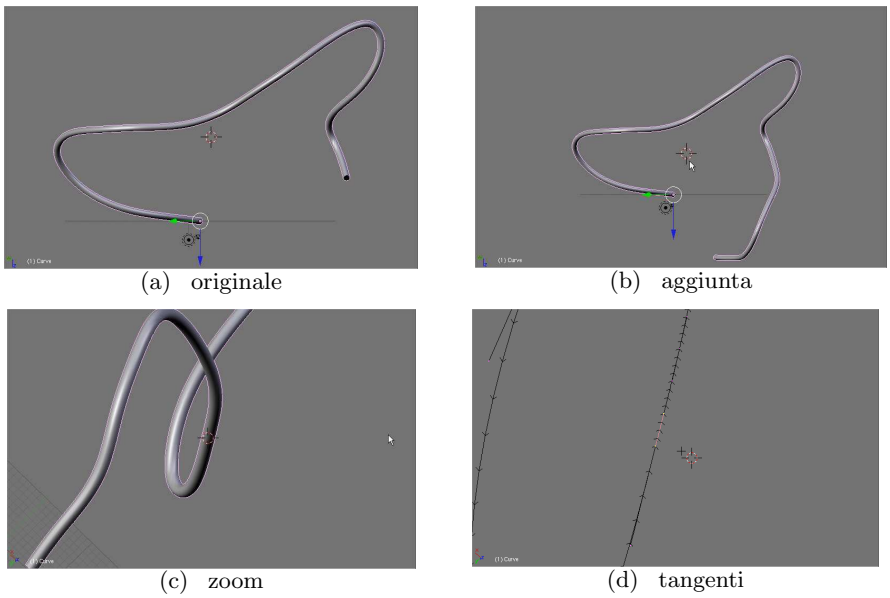
(c) zoom (d) tangenti

Figura 6.5: innesto di una curva su di un'altra

Figura 6.6: seconda immagine per il testing

Utilizzando l'immagine 6.6 sono stati effettuati vari inserimenti, in figura 6.7 è visualizzato un inserimento in coda.

Le immagini in figura 6.7 rappresentano un inserimento all'ultimo punto



(a) originale (b) aggiunta

(c) zoom (d) tangenti

Figura 6.7: innesto in coda di una curva su di un'altra

di una curva composta da venti punti con valori di α, β e γ pari a 0°, questo inserimento viene chiamato inserimento in coda. Nella terza immagine il punto di innesto è situato nelle stesse coordinate indicate dal cursore 3d. Come si può notare dalla figura 6.7, l'inserimento in coda permette di ottenere una curva continua rendendo impossibile il riconoscimento del punto di innesto.

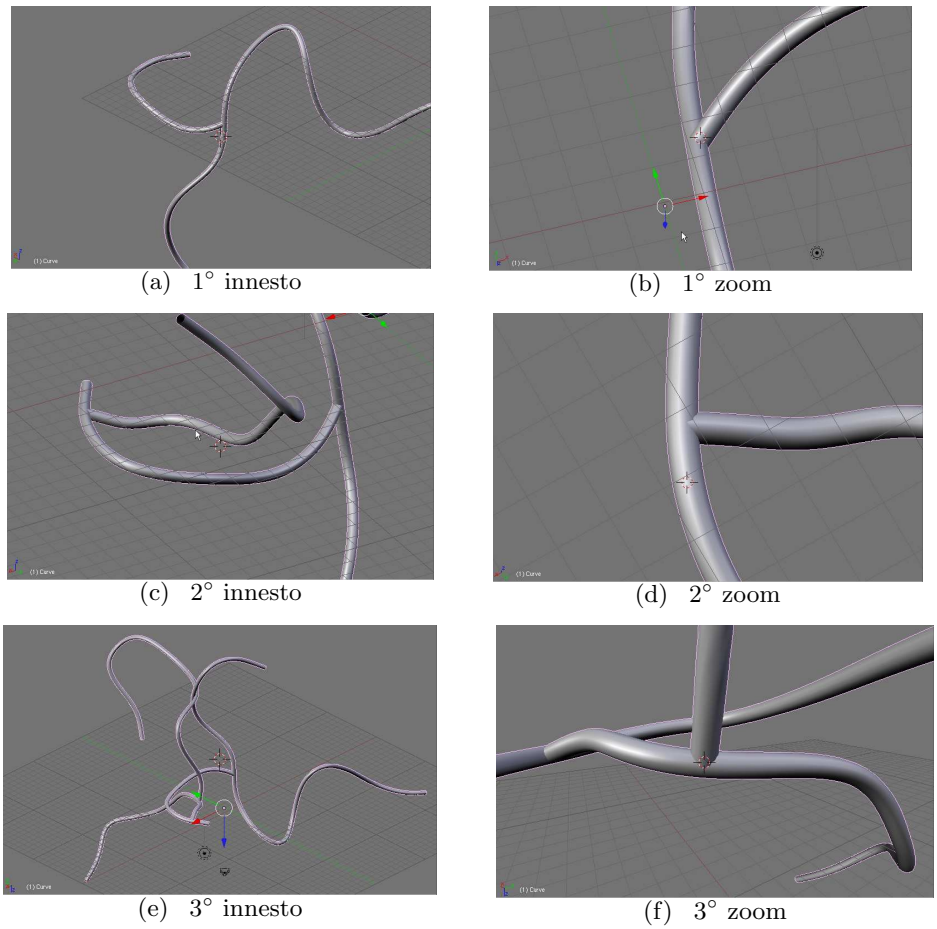
Utilizzando il software realizzato in questa sede è possibile creare strutture multi-curve aggiungendo a passi successivi curve alla struttura già presente. La figura 6.8 è un perfetto esempio di questo.

Gli innesti nell'immagine 6.8 sono realizzati con angoli differenti, in modo da testare separatamente l'effetto di α, β e γ. Tutti gli innesti sono fatti all'interno delle curve, non vi sono cioè inserimenti in testa o in coda. Inoltre, l'esempio di figura 6.8 serve a testare l'interazione di curve provenienti da più fonti, le curve innestate, infatti provengono sia dall'immagine 6.6 che dall'immagine 6.1. Rispettivamente tali immagini individuano:

1. una struttura a 2 curve. Il primo innesto ha valori di α, β e γ uguali a 45°, entrambe le curve provengono da 6.6;

2. lo zoom del primo innesto;

3. una struttura a 3 curve. Il secondo innesto assume per gli angoli i



(a) 1° innesto (b) 1° zoom

(c) 2° innesto (d) 2° zoom

(e) 3° innesto (f) 3° zoom

Figura 6.8: struttura multi-curva a più innesti consecutivi

valori di $\alpha = 90°$, $\beta = 0°$ e $\gamma = 0°$, la curva viene innestata sulla seconda curva creata e proviene da 6.1;

4. lo zoom del secondo innesto;

5. struttura a 4 curve. Il terzo innesto, infine, fissa gli angoli ai valori $\alpha = 0°$, $\beta = 90°$ e $\gamma = 0°$, la curva viene innestata sulla terza curva creata e proviene da 6.1;

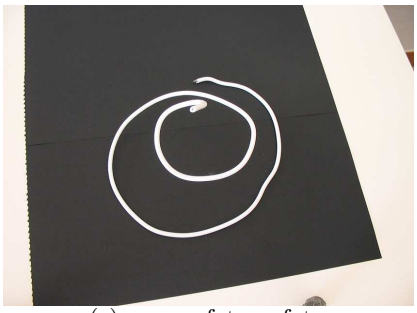
6. lo zoom del terzo innesto;

La fase di testing, infine, si chiude con la prova dello scatto da remoto della fotocamera con conseguente creazione della curva fotografata, immagine 6.9.

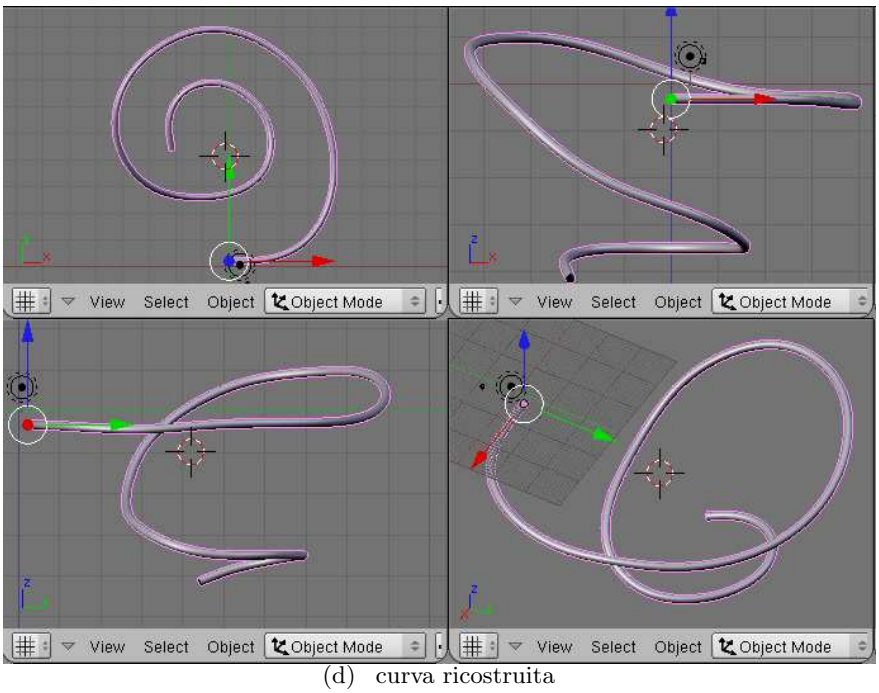
Le immagini in figura 6.9 rappresentano:

(a) ambientazione

(b) curva da ricostruire



(c) curva fotografata



(d) curva ricostruita

Figura 6.9: ricostruzione da fotografia scattata in remoto

1. La prima immagine identifica l'ambientazione in cui la fotografia viene scattata;
2. la seconda rappresenta il particolare della curva da ricostruire;
3. la terza fotografia è quella scattata da remoto da cui è stata ricostruita la curva 3d;
4. l'ultima immagine, infine, rappresenta la curva ricostruita da quattro diverse visuali.

6.2 Utilizzi della curva in Blender

Le curve in Blender offrono principalmente quattro utilizzi, ma un utente esperto di Blender può sicuramente conoscerne ulteriori:

1. come struttura fine a se stessa, avvolgendo la curva con una superficie generata tramite l'impostazione dei valori di estrusione;
2. come curva che definisce un percorso di un oggetto in un'animazione;
3. come superficie bidimensionale, racchiusa dalla curva stessa, che permette il disegno di loghi o comunque di curve figure geometriche bidimensionali;
4. si può usare una curva, inoltre, per deformare un oggetto.

Nel seguito vengono illustrati più dettagliatamente i quattro utilizzi elencati.

6.2.1 Utilizzo come struttura

Il primo, e più ovvio, modo di utilizzo di una curva è come oggetto fisico all'interno della scena.

Blender permette l'utilizzo di diversi parametri che identificano la sezione della struttura costruita attorno alla curva:

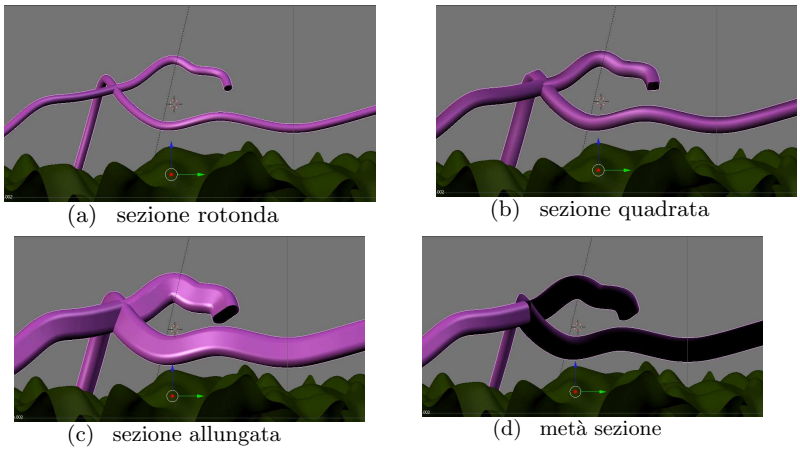
Extrusion : indica la lunghezza della sezione planare lungo la normale di curvatura, può variare da 0 a 5.

Bevel Depth : indica il raggio della sezione, se **Extrusion** è diverso da 0 indica il raggio della curva agli estremi della faccia planare della sezione definita dall'estrusione, varia da 0 a 2.

Bevel Resolution : indica il numero di segmenti che deve comporre la parte curva della sezione, varia da 0, che identifica 4 segmenti, a 32, che identifica 68 segmenti.

In aggiunta a questi parametri vi sono due toggle che identificano la forma della struttura che viene creata attorno alla curva: i toggle **back** e **front**. Se entrambi i toggle non sono selezionati, la struttura risultante avvolge per intero la curva e la sezione risulta simmetrica, altrimenti appare solo metà sezione.

In figura 6.10, appaiono quattro esempi di sezioni attorno alla curva.



(a) sezione rotonda (b) sezione quadrata

(c) sezione allungata (d) metà sezione

Figura 6.10: quattro esempi di sezioni

Rispettivamente le quattro sezioni rappresentate individuano:

1. nella prima i valori impostati per le tre impostazioni sono: **Extrusion** uguale a 0, **Bevel Depth** uguale a 1 e **Bevel Resolution** uguale a 5. La sezione risultante è rotonda. Si noti che questi sono i valori preimpostati alla creazione della curva tramite l'applicativo ivi trattato.

2. nella seconda i valori impostati sono: **Extrusion** uguale a 0, **Bevel Depth** uguale a 2 e **Bevel Resolution** uguale a 0. La sezione risultante è quadrata.

3. nella terza i valori impostati sono: **Extrusion** uguale a 3, **Bevel Depth** uguale a 2 e **Bevel Resolution** uguale a 10.

4. infine, nella quarta i valori impostati sono gli stessi della precedente con la differenza che viene selezionato il toggle **front**.

In appendice C è possibile visualizzare esempi di utilizzo come struttura in C.1, C.2, C.5 e C.6.

6.2.2 Utilizzo come cammino

Il secondo utilizzo adoperabile consiste nello strutturare la curva come un cammino percorribile da un oggetto.

La possibilità di seguire una curva 3d è probabilmente l'utilizzo di maggiore interesse delle curve in Blender, ciò permette di gestire traiettorie anche complesse semplicemente disegnando una curva, o, nel caso dell'applicativo qui discusso, semplicemente piegando un tubicino.

Per ottenere questo risultato, innanzitutto, bisona impostare due toggle e una impostazione della curva:

CurvePath : toggle che, se impostato, individua la curva come cammino per animazioni.

CurveFollow : toggle che, se impostato, obbliga l'oggetto assegnato alla curva a essere ruotato con la direzione della tangente della curva in ogni suo punto.

PathLen : impostazione numerica che individua la lunghezza, in frame, del cammino sulla curva.

A questo basta aggiungere ad un oggetto il vincolo di seguire il cammino identificato dalla curva e il movimento nell'animazione è completato.

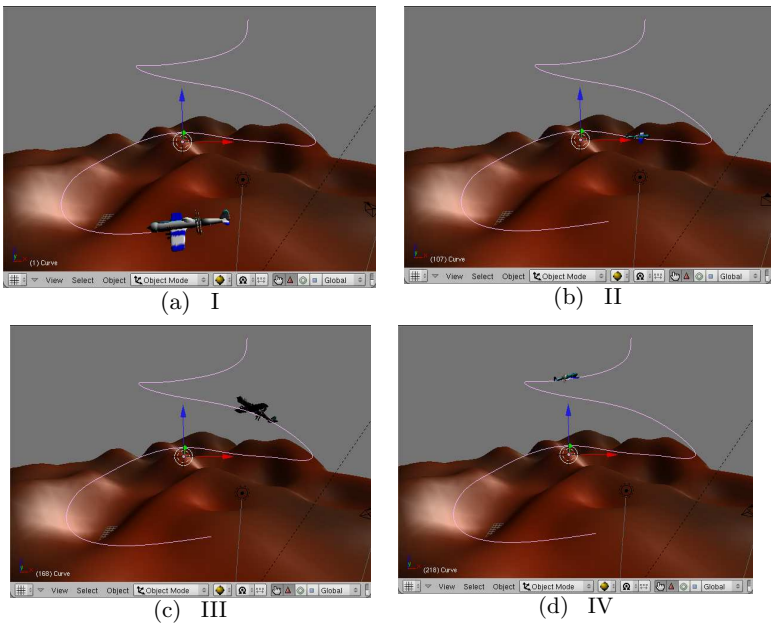
In figura 6.11 è mostrato un esempio di inseguimento della curva estratta in figura 6.9. Questo cammino è definito impostando entrambi i toggle precedentemente definiti.

In figura 6.12, invece, è mostrato un esempio, sempre riferito alla figura 6.9, in cui il toggle **CurveFollow** non è impostato. Come si può vedere l'oggetto non ruota seguendo la tangente della curva, ma rimane fisso e semplicemente trasla.

Il rendering dell'esempio sopra riportato è visualizzabile in appendice C, in figura C.9. Altri esempi di inseguimento della curva sono i video nelle figure C.7 e C.8.

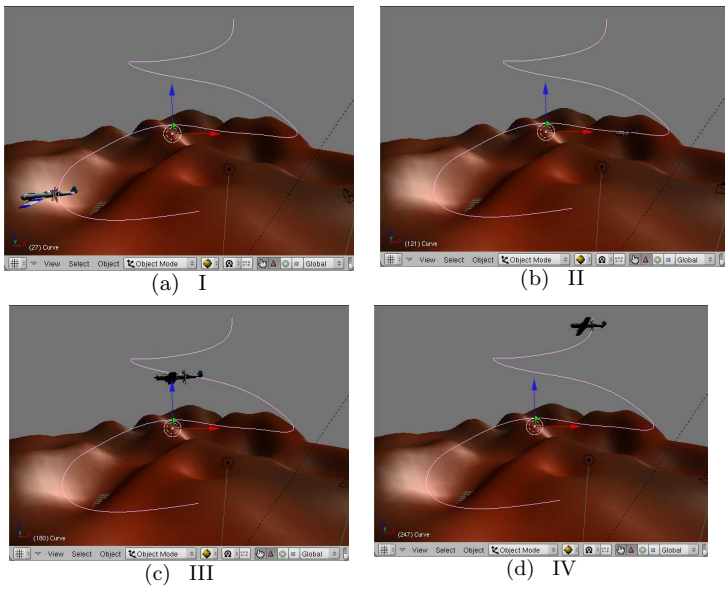
6.2.3 Utilizzo come superficie 2d per loghi

Originariamente le curve in Blender sono bidimensionali; per renderle 3d è necessario attivare un toggle apposito chiamato **3d**. Questo perchè le curve



(a) I (b) II (c) III (d) IV

Figura 6.11: esempio di inseguimento della curva



(a) I (b) II (c) III (d) IV

Figura 6.12: secondo esempio di inseguimento della curva

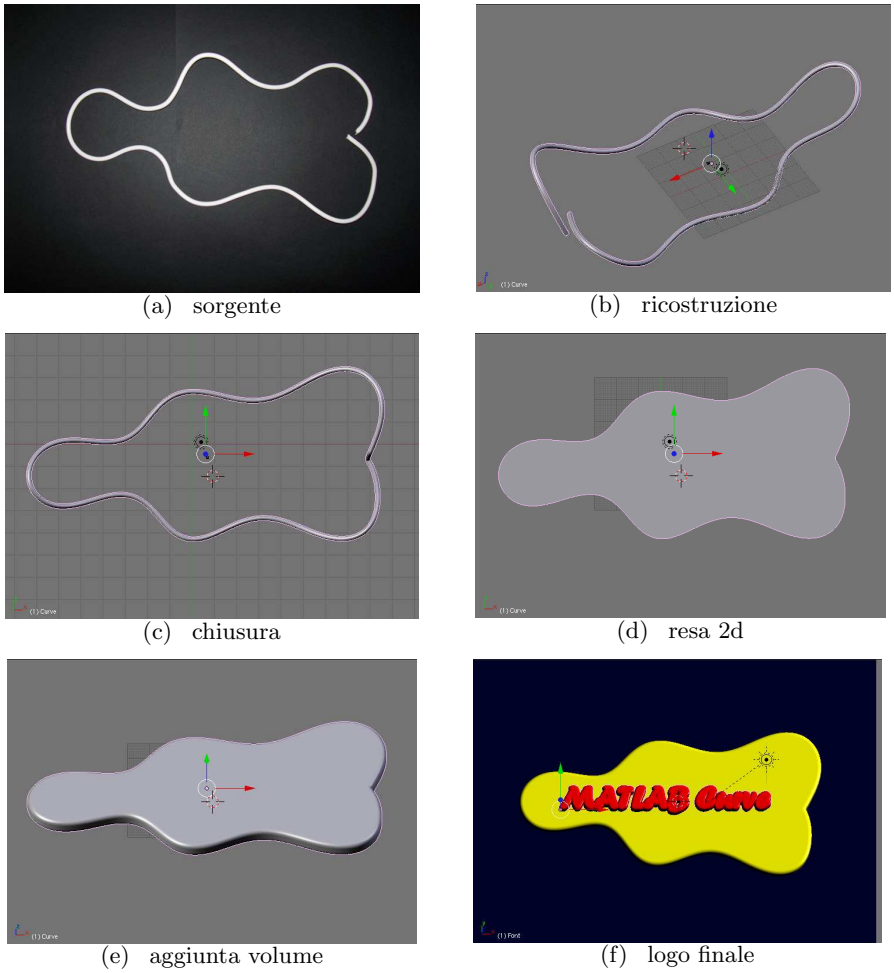
tridimensionali sono difficilmente manipolabili all'interno di Blender senza

strumenti aggiuntivi quali quello sviluppato per questa tesi.

Da questo deriva che uno degli sviluppi primari per cui le curve sono utilizzati in Blender è la creazione di loghi[1] con l'utilizzo di superfici 2d.

Per ottenere una superficie basta chiudere una curva 2d con i comandi appositi.

Nella figura 6.13 è mostrato un procedimento esemplificativo per la creazione di un logo.



(a) sorgente (b) ricostruzione

(c) chiusura (d) resa 2d

(e) aggiunta volume (f) logo finale

Figura 6.13: creazione di un logo

La costruzione di un logo si articola nei passi seguenti:

[1] per logo si intende una figura piana bidimensionale, con o senza scritte, che identifichi il simbolo di qualcosa.

1. Prima di tutto si ottiene la curva dall'immagine tramite l'applicativo prodotto per questa tesi, immagini a e b della figura 6.13.

2. Poi si chiude la curva e la si rende bidimensionale deselezionando il toggle chiamato **3d**, immagini c e d. Questa operazione di resa bidimensionale proietta la curva sul piano xy e su quel piano ottiene la superficie.

3. Successivamente si impostano i valori desiderati per **Extrusion**, **Bevel Resolution** e **Bevel Depth** il cui comportamento è molto simile al caso precedentemente illustrato della sezione 6.2.1, immagine e. I toggle **front** e **back**, invece, assumono un'altro significato:

   (a) se entrambi, o il solo **front**, sono selezionati, al variare dei sopracitati parametri la superficie si "rigonfia "sull'asse z formando una superficie convessa.

   (b) se il solo **back** è selezionato, allora al variare dei parameteri la superficie si "svuota "sull'asse z formando una superficie concava.

   (c) se nessuno dei due è selezionato, non viene individuata la superficie, bensì il contorno , che viene strutturato come nel caso in sezione 6.2.1, solo che la curva giace su un piano.

4. Infine, si aggiungono vari effetti, quali materiali, luci e scritte e si ottiene un logo completo, immagine f.

In appendice C viene mostrato un altro esempio di logo nella figura C.3.

6.2.4 Deformazione di oggetti tramite curva

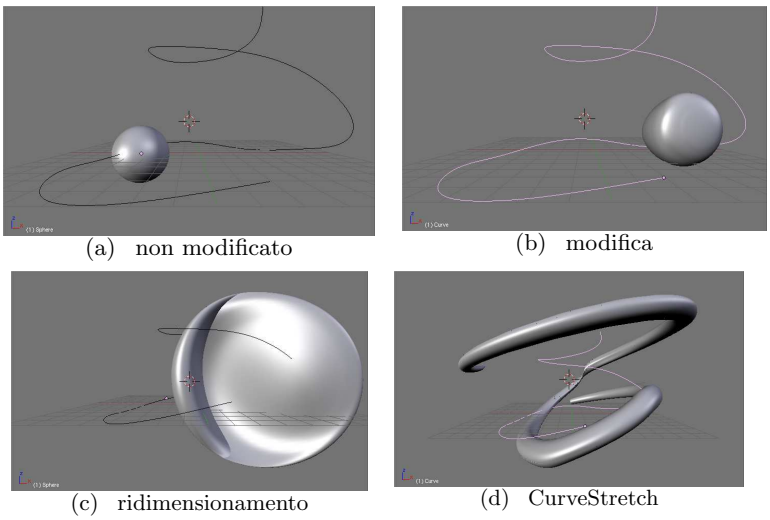
L'ultimo utilizzo qui presentato riguarda la deformazione di oggetti secondo l'andamento di una curva.

Tramite un modificatore per oggetti è possibile, infatti, modificare la forma di un oggetto utilizzando semplicemente una curva.

Il modificatore trasforma l'oggetto originale modificando le coordinate dei punti appartenenti all'oggetto in funzione dei punti della curva. Non è ben chiaro secondo quali leggi il procedimento lavori, ma gli effetti sono comunque visibili in figura 6.14

La curva possiede un toggle relativo alle modalità di modifica degli oggetti:

CurveStretch : se selezionato impone all'oggetto modificato di seguire nella sua forma l'intera curva, l'effetto è visibile nell'immagine d della figura 6.14.



(a) non modificato (b) modifica

(c) ridimensionamento (d) CurveStretch

Figura 6.14: esempi di trasformazione seguendo la curva

Le immagini in figura 6.14 rappresentano rispettivamente:

1. l'oggetto non modificato, in questo caso una sfera;
2. l'oggetto modificato dalla curva;
3. l'oggetto ingrandito e modificato dalla curva;
4. l'oggetto che viene modificato dalla curva nel suo insieme.

In appendice C è visualizzabile un esempio renderizzato nrlla figura C.4

6.3 Valutazione del lavoro svolto

Finora si è discusso del funzionamento dell'intero sistema e dei suoi dettagli concettuali, ora è necessario valutarne le prestazioni e identificare i punti deboli del programma.

Le prestazioni del sistema sono dominate e determinate dall'algoritmo di elaborazione dell'immagine e successiva ricostruzione della curva, che rappresenta la parte computazionalmente più complessa dell'intero processo. Questo procedimento, sebbene piuttosto dispendioso di risorse, non può però essere migliorato, in quanto i calcoli stessi da svolgere per ricostruire la curva risultano pesanti da compiere.

In data odierna, il software di ricostruzione della curva presenta una pesante limitazione: non è possibile ricostruire curve con parti sovrapposte; se

la curva si incrocia in qualche modo su se stessa non è possibile ricostruirla per intero. Per ricostruire correttamente la curva è quindi necessario che la curva inquadrata della fotocamera non presenti sovrapposizioni. Comunque la soluzione a questo problema è in via di sviluppo.

Un aspetto migliorabile dell'applicativo prodotto risiede nell'interfaccia utente in Blender per l'innesto di curve su altre curve; ritengo particolarmente scomoda la selezione del punto di innesto sulla curva tramite un numero di ordinamento. Sarebbe auspicabile, invece, selezionare il punto sulla curva manualmente dalla scena 3d, potendo così visualizzare in via preventiva il punto in cui la seconda curva verrà innestata. Questa ottimizzazione richiede, però, conoscenze più approfondite delle API fornite da Blender; potrà comunque essere implementata in uno sviluppo futuro del progetto.

Per tutti gli altri aspetti il software riesce a soddisfare completamente gli obbiettivi che ci si era preposti per questo lavoro.

6.4 Conclusioni del capitolo

Nel presente capitolo è stata mostrata la fase di testing del sistema e si è introdotto il lettore nelle possibilità creative che il software offre.

Sono stati, inoltre, discussi i punti deboli del programma creato, fornendo alcune idee per migliorarlo.

Nel prossimo capitolo si discuterà degli sviluppi futuri del campo di ricerca trattato e si trarranno le conclusioni.

Capitolo 7

Direzioni future di ricerca e conclusioni

RIASSUMENDO, il lavoro svolto rappresenta un'interfaccia tangibile per il disegno di curve spaziali in un software di modellazione 3D, Blender. L'ambito delle interfacce tangibili rappresenta la creazione di strumenti fisici per la ricostruzione di oggetti 3d, per semplificare il processo di modellazione, o la navigazione 3d. Il mouse e la tastiera, infatti, non rappresentano strumenti ottimali o efficenti per lavorare con complesse strutture 3d, in quanto, il mouse manca di sufficenti gradi di libertà, mentre la tastiera non presenta un funzionamento comodo o immediato per la navigazione in ambienti 3d. Si è cosi reso necessario lo studio di nuovi strumenti per facilitare la creazione o la modellazione di oggetti 3d. Le soluzioni proposte dalla ricerca si suddividono in due categorie:

- strumenti a 6 DOF[1], quali, ad esempio, mouse a più gradi di libertà.
- le interfacce tangibili quali strumenti studiati ad hoc per risolvere alcuni sottoproblemi. Per la loro specificità le interfacce tangibili rappresentano senza alcun dubbio una soluzione ottimale, rispetto a strumenti non studiati appositamente, del problema che si pongono di risolvere.

L'area di inquadramento di questa tesi comprende anche la tecnica di ricostruzione 3d chiamata *shape from contour*, in quanto il cuore del sistema realizzato, l'algoritmo di ricostruzione della curva 3d partendo da singola immagine, è un esempio di *shape from contour*.

[1]Degree of Freedom, gradi di libertà in inglese

L'intento del presente lavoro è, quindi, quello di realizzare un'interfaccia che, utilizzando una tecnica di *shape from contour*, fornisca all'utente un dispositivo comodo per riprodurre curve nello spazio 3d manipolando un oggetto fisico, nello specifico un tubicino. Questo comportamento è stato ottenuto integrando diversi software, e fornendo all'utente alcune funzionalità per la manipolazione della curva ricostruita. Nello specifico si è provveduto a creare ed integrare vari moduli che coordinati permettessero di ottenere il comportamento desiderato.

Il software funziona realizzando alcuni passaggi:

scatto da remoto di una fotografia da fotocamera digitale : questo modulo, realizzato grazie all'utilizzo di uno specifico toolbox per matlab, Camera Box [24], permette di ottenere un'immagine da fotocamera digitale in modo automatico dal computer, senza dover manualmente scattare la fotografia. Questa feature permette all'utente, dopo che è stato impostato l'ambiente fisico dove scattare la fotografia, di lavorare su curve 3d consecutivamente manipolando il tubicino sotto l'obiettivo della fotocamera e scattando da remoto la fotografia, senza dover continuamente reimpostare il tutto.

ricostruzione della curva 3D : una volta ottenuta la fotografia, utilizzando un algoritmo di ricostruzione 3d pre-esistente scritto in Matlab [25], si procede a ricostruire la curva 3d sotto forma di spline. Successivamente, si trasforma la curva ricostruita in formato Bézier e la si rototrasla secondo i valori passati in input dall'interfaccia grafica di Blender, da cui l'applicativo in questa sede trattato viene lanciato. Questa parte è completamente scritta in Matlab.

comunicazione tra Matlab e Blender : si è reso necessario a questo punto coordinare i due strumenti software utilizzati, Matlab e Blender, tramite un wrapper di Matlab per Python, chiamato MatlabWrap. Questa libreria permette di richiamare funzioni Matlab dallo script Python che, tramite opportune API fornite da Blender, estende le funzionalità del programma di modellazione 3d.

disegno della curva in Blender : una volta ottenuta la curva in Blender, nel formato giusto, cioè Bézier, è possibile disegnarla sfruttando il motore grafico fornito dal software. Questo avviene tramite le librerie apposite che permettono di modificare automaticamente gli oggetti presenti nella scena di Blender. Ciò ha permesso di aggiungere un'interessante funzionalità all'applicativo prodotto: la possibilità di

innestare più curve una sull'altra, in modo da ottenere strutture anche molto complesse ed aumentare così l'utilità dell'applicativo realizzato.

interfaccia utente : infine, sempre utilizzando le librerie offerte da Blender, è stata realizzata una semplice, ma efficace, interfaccia utente, che permette a quest'ultimo di sfruttare le potenzialità offerte dal software prodotto.

Il lavoro qui presentato rappresenta uno strumento efficace per la creazione di curve tridimensionali. Inoltre, l'utilizzo di Blender permette di sfruttare tutte le potenzialità offerte dal software stesso: è, infatti, possibile usare la curva ricostruita in svariati modi:

- per renderizzare tubi di svariate dimensioni o materiali;
- per usare le curve come cammini che possano definire il movimento degli oggetti associati;
- per creare loghi complessi, disegnando superfici bidimensionali;
- per deformare oggetti secondo la forma della curva.

Dopo aver richiamato tutti gli aspetti del lavoro svolto, passiamo ora a proporre degli ulteriori sviluppi del progetto.

Per proseguire il progetto può essere utile rendere compatibile il software con le superfici NURBS di Blender, che permettono di costruire superfici 3d partendo da curve 3d o 2d.

Altrimenti si possono automatizzare alcuni processi, in modo da creare dei rendering con alcune caratteristiche, ad esempio rappresentazione di montagne russe virtuali, o altre feature il cui limite è imposto solo dalla fantasia del programmatore.

Oppure è possibile sfruttare il game engine di Blender e creare un qualche tipo di videogioco con il software prodotto, ad esempio creando un videogioco di corse con navicelle dove il percorso viene definito automaticamente dalla curva ricostruita.

Sarebbe interessante, inoltre, provare a realizzare un'interfaccia tangibile per strutture 3d complesse, ad esempio a forma di omino composto di tubicini come quello in questo software realizzato. Ciò potrebbe essere ottenuto sfruttando il medesimo algoritmo usato in questa sede, ottimizzato per la gestione di tubi sovrapposti.

Concludendo, le interfacce tangibili, sebbene strettamente vincolate alla risoluzione di problemi specifici, rappresentano un'opportunità di studio approfondito e mirato dell'interazione Uomo-Macchina. Applicazioni come

quella in questa sede sviluppata, sono sicuramente validi strumenti atti a semplificare la vita dell'utente. Più nello specifico tutti gli artisti del 3D, quali designer, pubblicitari e autori di videogiochi o film di animazione 3d, possono trovare un efficace aiuto al loro lavoro nel software in questa sede prodotto: MATLAB Curve.

Bibliografia

[1] Gianfranco Zosi, *Spline*, http://newton.ph.unito.it/žosi/f2-analdati/pdf-analdati/spline.pdf

[2] Dr. Thomas Sederberg, *BYU Bézier curves*, http://www.tsplines.com/resources/class_notes/Bezier_curves.pdf

[3] Les Piegl - Wayne Tiller, *The NURBS Book*, Springer-Verlag, 1995-1997

[4] Shumin Zhai, *Sketching in 3D*, http://www.siggraph.org/publications/newsletter/v32n4/contributions/zhai.html

[5] Ali Mazalec - Michael Nitsche, *Tangible Interfaces for Real-Time 3D Virtual Environments*, Georgia Institute of Technology, Atlanta

[6] Michele Fiorentino - Giuseppe Monno - Antonio E. Uva, *Manipolazione interattiva di curve e superfici in ambiente virtuale*, Politecnico di Bari

[7] Steven Schkolne - Hiroshi Ishii - Peter Schröder, *Tangible + Virtual = A Flexible 3D Interface for Spatial Construction Applied to DNA*, MIT Media Lab

[8] Hyosun Kim - Georgia Albuquerque - Sven Havemann - Dieter W. Fellner*Tangible 3D: Immersive 3D Modeling through Hand Gesture Interaction*, Institute of Computer Graphics University of Technology Mühlenpfordtstr

[9] Fukutake Hiromichi - Okada Yoshihiro - Niijima Koichi, *Voice Input Interface for Development of 3D Graphics Software*, IEIC Technical Report (Institute of Electronics, Information and Communication Engineers)

[10] Karin Leichtenstern - Elisabeth André - Eva Lösch - Matthias Kranz - Paul Holleis, *A Tangible User Interface as Interaction and Presentation Device to a Social Learning Software*, University of Augsburg - University of Munich

[11] Matthias Kranz - Dominik Schmidt - Paul Holleis - Albrecht Schmidt, *A Display Cube as a Tangible User Interface*, Research Group Embedded Interaction, Munich

[12] Brygg Ullmer - Hiroshi Ishii, *The metaDESK: Models and Prototypes for Tangible User Interfaces*, MIT Media Lab, Cambridge

[13] David Anderson - James L. Frankel - Joe Marks - Aseem Agarwala - Paul Beardsley - Jessica Hodgins - Darren Leigh - Kathy Ryall - Eddie Sullivan - Jonathan S. Yedidia, *Tangible Interaction + Graphical Interpretation: A New Approach to 3D Modeling*, MERL-Mitsubishi Electric Research Laboratory: University of Virginia - Georgia Institute of Technology

[14] Eleonora Bilotta, *Interfacce multimediali ed aspetti psicologici dell'interazione uomo-computer*, capitolo sesto, Editoriale BIOS, http://galileo.cincom.unical.it/Pubblicazioni/editoria/libri/HCI-ele/cap6.html

[15] Ruo Zhang - Ping-Sing Tsai - James Edwin Cryer - Mubarak Shah, *Shape from Shading: A Survey*, University of Central Florida

[16] Emmanuel Prados - Olivier Faugeras, *Shape From Shading*

[17] Ari D. Gross, *Shape from contour methods using object-based heuristics*, AA(CUNY/Queens College and Columbia Univ.)

[18] Alexander Belyaev, *Gaussian Curvature. Shape from Contour. Special Surfaces*

[19] Helmut Cantzler, *An overview of shape-from-motion*

[20] Canon,*Digital Imaging Developer Programme*, http://www.didp.canon-europa.com/

[21] Blender Community, *Blender wiki*, http://wiki.blender.org/index.php/Main_Page

[22] The MathWorks, *Matlab documentation*, http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml

[23] Anrew Davinson, *Pro Java 6 3D Game Development*, Apress, http://fivedots.coe.psu.ac.th/ãd/jg2/

[24] Sean O'Malley,*CameraBox v1.30, Canon® Camera Control Toolbox for Matlab®*, http://www2.cs.uh.edu/s̃omalley/camerabox.html

[25] Alessandro Giusti,*Reconstruction of canal surfaces from single perspective images*,tesi di laurea del politecnico di milano aa 2004-2005

[26] Wikipedia,*TNB Frame*, url=http://en.wikipedia.org/wiki/TNB_Frame

[27] Alexander Schmolck, *mlabwrap v-1.0*, http://mlabwrap.sourceforge.net/

[28] The MathWorks, *The MATLAB engine*, http://www.mathworks.com/access/helpdesk/help/techdoc/index.html ?/access/helpdesk/help/techdoc/matlab_external/f29148.html

[29] Guido van Rossum, *Python documentation*, http://www.python.org/

[30] The Blender Python Team, *The Blender Python API Reference*, http://www.blender.org/documentation/245PythonDoc/index.html

Appendice A

Listato

In questa appendice sono stati inseriti i sorgenti degli script matlab e Blender.

Convenzione:

- i commenti si riferiscono alle righe di codice successive o immediatamente a lato

A.1 Script matlab

Qui di seguito sono inseriti i due script matlab:

1. **getPhoto.m**: codice per eseguire lo scatto da fotocamera via remoto.
2. **getCurve.m**: codice per l'elaborazione dell'immagine e la ricostruzione della curva.

A.1.1 getPhoto.m

```
function [filepath]=getPhoto(path,flash)

% Funzione che permette lo scatto in remoto da
% fotocamera canon.
%
% SINTASSI:
%       [filepath] = getPhoto(path,flash)
%
% PARAMETRI
%       path        directory in cui si vuole salvare
%                   l'immagine scattata
%       flash       booleano che imposta o meno il flash
%
```

```
% OUTPUT
%       filepath  path in cui si è effettivamente
%                 salvato il file; se coincide con
%                 path la funzione è terminata
%                 correttamente, altrimenti si è
%                 verificato un errore e assume
%                 valore 'error'
%
% CONVENZIONE
%       Richiamando la funzione con un path valido e
%       se è correttamente connessa via USB una
%       fotocamera Canon compatibile con le SDK viene
%       scattata la fotografia in remoto e salvata
%       nel path in input.
%

%% cambio della directory corrente

    % si cerca la current directory
    curr=cd();
    % la si cambia in curr\Cambox130
    cd(strcat(curr,'\CamBox130'));

%% uso di Camera Box

    % si inizializza il valore dell'output a error
    filepath='error';

    % si inizializza Camera Box
    cc=InitCameraBox;
    % si controlla se ci sono stati errori
    [iserr,errstr] = CB_IsError(cc);
    if (iserr)
        fprintf('\nError! "%s"\n', errstr);
        % si reimposta la cd a quella iniziale
        cd(curr);
        % si termina la funzione
        return
    end

    % si cercano le fotocamere  connesse
    camNames = CB_GetConnectedCameraNames(cc);
    [iserr,errstr] = CB_IsError(cc);
    if (iserr)
        fprintf('\nError! "%s"\n', errstr);
        % si termina CameraBox
        DestroyCameraBox(cc);
        cd(curr);
        return
```

```
end


Done=false; % bool che indica lo scatto della foto

% si ricerca fra tutte le fotocamere connesse
for i=1 : length(camNames)

    % ci si connette all'iesima fotocamera
    CB_OpenSource(cc, i);
    [iserr,errstr] = CB_IsError(cc);
    if (iserr)
        % si passa alla prossima fotocamera
        continue
    end

    % si passa a modalità scatto
    CB_EnterReleaseControl(cc);
    [iserr,errstr] = CB_IsError(cc);
    if (iserr)
        % ci si disconnette dalla fotocamera
        CB_CloseSource(cc);
        continue
    end

    % si controlla il flag del flash
    if flash==true
        % si ottiene le impostazioni possibili
        [x y]=CB_GetFlashSettings(cc);
        [iserr,errstr] = CB_IsError(cc);
        if (iserr)
            % si esce dalla modalità scatto
            CB_ExitReleaseControl(cc);
            CB_CloseSource(cc);
            continue
        end
        % si imposta il flash dove per Canon A60
        % 2= flash normale
        % 10= 0 correzione esposizione
        CB_SetFlash(cc, x, 2, y, 10);
    end

    % si scatta la foto e salva in path
    CB_Release(cc, path);
    [iserr,errstr] = CB_IsError(cc);
    if (iserr)
        CB_ExitReleaseControl(cc);
        CB_CloseSource(cc);
        continue
```

```
        end

        % si indica lo scatto della foto
        Done=true;

        % si esce dalla modalità scatto
        CB_ExitReleaseControl(cc);

        % si disconnette la fotocamera
        CB_CloseSource(cc);

        % si termina cameraBox
        cc=DestroyCameraBox(cc);
    end

%% terminazione funzione

    % si reimposta la cd a quella iniziale
    cd(curr);

    % si controlla se è stato fatto lo scatto
    if Done
        % si scrive correttamente l'output
        filepath=path;
    end
end
```

A.1.2 getCurve.m

```
function [vert]=getCurve(path,number,point,alpha,beta,gamma,tang
  ,binorm)

% Funzione che permette la ricostruzione di una curva 3d
% partendo da una singola immagine rappresentante una canal
% surface. Fornisce inoltre strumenti per manipolarla a
% piacere.
%
% SINTASSI:
%       [vert]=getCurve(path,number,point,alpha,beta,gamma,tang,
  binorm)
%
% PARAMETRI
%       path    percorso del file immagine da elaborare
%       number  numero di punti della curva ricostruita
%       point   vettore 3d identificante il punto origine
%               della curva
%       alpha   angolo sul piano xy che identifica la
%               tangente iniziale della curva
%       beta    angolo sul piano xz che identifica la
```

```
%                tangente iniziale della curva
%        gamma   angolo sul piano yz che identifica la
%                normale iniziale della curva
%        tang    vettore 3d che identifica l'asse x del
%                sistema di riferimento della curva
%        binorm  vettore 3d che identifica l'asse z del
%                sistema di riferimento della curva
%
% OUTPUT
%        vert    matrice 9Xn, dove n è il numero di punti
%                della curva composta da: le prime tre
%                colonne identificano la tangente sx del
%                punto, le tre centrali identificano le
%                coordinate del punto stesso, le ultime 3
%                identificano la tangente dx del punto.
%
% CONVENZIONE
%        I tre vettori 3d devono essere 1x3.
%        Se l'immagine identificata da path contiene una
%        canal surface (un tubo), la funzione restituisce
%        la curva 3d rapppresentante quel tubo nel sistema
%        di riferimento definito da tang e binorm, con punto
%        iniziale point e direzione definita da alpha, beta
%        e gamma. La curva restituita è nella forma di curva
%        Bézier.
%

%% valutazione del numero degli argomenti

% se i valori in input sono 2 si mantiene il sistema
% di riferimento originale della curva, altrimenti
% si calcola il nuovo sistema di riferimento
if nargin==2
    modcoord = 0;
else
    modcoord = 1;
end

%% Loading image
[...]

%% Extracting contours
[...]

%% Fitting splines
[...]

%% Draw borders
[...]
```

```
%% Reconstruction
[...]

%% Fitting spline to axis

% calcolo differenze nei punti ricostruiti
df = diff(centers);
% creazione ascissa curvilinea
t = cumsum([0, sqrt([1 1 1]*(df.*df)')]);
% creazione spline iniziale
[spine,r] = csaps(t,centers');
% Smooth more than default & calculate newx and newy
newx = linspace(t(1),t(end),number);
newy = csaps(t,centers',r/splinesmooth,newx);
% calcolo della spline con il numero di punti definito da number
spline = csaps(newx,newy,1);

%% converting spline into Bézier

% estrazione parametri dalla spline
vert = zeros(size(spline.breaks,2),9);
coef = spline.coefs;

% applicazione sistema di trasformazioni da spline a Bézier
for i=1:size(vert,1)-1
    d = (spline.breaks(i+1)-spline.breaks(i));
    if i==1
        vert(1,1:3) = coef(1:3,4);
        vert(1,4:6) = coef(1:3,4);
    end
    vert(i,7:9) = coef(3*(i-1)+(1:3),3)*d/3+coef(3*(i-1)+(1:3)
      ,4);
    vert(i+1,1:3) = coef(3*(i-1)+(1:3),2)*d^2/3+coef(3*(i-1)
      +(1:3),3)*2*d/3+coef(3*(i-1)+(1:3),4);
    vert(i+1,4:6) = coef(3*(i-1)+(1:3),1)*d^3+coef(3*(i-1)+(1:3)
      ,2)*d^2+coef(3*(i-1)+(1:3),3)*d+coef(3*(i-1)+(1:3),4);
    if i==size(vert,1)-1
        vert(i+1,7:9) = vert(i+1,4:6);
    end
end

%% apply rototranslation

if modcoord==1
    % trasformazione angoli in radianti
    alpha = mod(alpha,360);
    beta = mod(beta,360);
    gamma = mod(gamma,360);
```

```
    alpha = alpha*2*pi/360;
    beta = beta*2*pi/360;
    gamma = gamma*2*pi/360;

    % traslazione curva nell'origine (0,0,0)
    newpoint = [-vert(1,4) -vert(1,5) -vert(1,6)];
    for i=1:size(vert,1)
        for j=1:3
            for k=0:2
                vert(i,3*k+j) = vert(i,3*k+j)+newpoint(j);
            end
        end
    end

    % normalizzazione tang e binormale in input
    tang = tang/norm(tang);
    binorm = binorm/norm(binorm);

    % calcolo derivata prima e seconda della spline
    splined = fnder(spline);
    splined2 = fnder(splined);
    funcder = fnval(splined,spline.breaks(1)+eps);
    funcder2 = fnval(splined2,spline.breaks(1)+eps);

    % calcolo TNB frame della curva nell'origine
    tangente = funcder/norm(funcder);
    binormale = cross(funcder,funcder2)/norm(cross(funcder,
      funcder2));
    normale = cross(binormale,tangente);

    % applicazione 3 rototraslazioni
    trans = [[tang;0] [cross(binorm,tang);0] [binorm;0] [point
      ;1]];
    trans = trans*[cos(alpha) -sin(alpha) 0 0;sin(alpha) cos(
      alpha) 0 0;0 0 1 0;0 0 0 1]*inv([cos(beta) 0 sin(beta) 0;0
       1 0 0; -sin(beta) 0 cos(beta) 0;0 0 0 1])*[1 0 0 0; 0 cos
      (gamma) -sin(gamma) 0; 0 sin(gamma) cos(gamma) 0;0 0 0 1];
    trans = trans*inv([[tangente;0] [normale;0] [binormale;0]
      [0; 0; 0; 1]]);

    % costruzione della matrice in output
    vertsx = trans*[vert(:,1:3)'; ones(1,size(vert,1))];
    vertctr = trans*[vert(:,4:6)'; ones(1,size(vert,1))];
    vertdx = trans*[vert(:,7:9)'; ones(1,size(vert,1))];

    vert = [vertsx(1:3,:)' vertctr(1:3,:)' vertdx(1:3,:)'];
end
```

A.2 Script Python per Blender

In questa sezione sono inseriti gli script python per blender:

1. **matCurveFunct.py**: modulo python per la gestione delle funzioni utili alla gestione della comunicazione matlab-blender e utili alla costruzione di curve in Blender

2. **MatCurve.py**: modulo python contenente la classe matCurve che permette di ottenere una curva dalla funzione matlab getCurve

3. **main.py**: main dello script python gestisce l'interfaccia utente in Blender, richiama tutte le funzioni necessarie al funzionamento di MATLAB Curve;

4. **setup.py**: consente di impostare la path corretta all'interno del file matCurveFunct.py senza dover modificare codice a mano.

A.2.1 matCurveFunct.py

```
##############################################
## MATLAB Curve: function library           ##
## ------------------                       ##
## Modulo per la gestione delle funzioni    ##
## necessarie a far funzionare lo script    ##
## correttamente, viene gestita la          ##
## comunicazione matlab-python              ##
##                                          ##
## Author: Lorenzo "A." Mureddu             ##
##############################################

# importazione librerie di sistema
import os
import time

# tentativo di importazione matlabwrap
try:
        from mlabwrap import mlab
        from numpy    import array
except:
        print("ERRORE impossibile trovare le librerie mlabwap o
          numpy prego installarle")

# importazione librerie blender
from Blender import Curve, BezTriple

# variabile globale def path del prog
```

```
SCRIPTPATH = r'C:\tesi\matlabCurve'

# serie di funzioni per gestire la curva matlab

# trasforma la lista di numeri vect in BezTriple
def bezFromVect(vect):
        """Trasforma una lista numerica in BezTriple

        Trasforma il vettore, sotto forma di lista,
        nel formato BezTriple di Blender caratterizzato
        da 9 float, 3 per tang sx, 3 per punto, 3 per
        tangente dx

        INPUT
        -vect lista contenente 9 numeri

        OUTPUT
        -k    Beztriple corrispondente a vect
        """
        k=BezTriple.New(vect)

        # imposta le maniglie dei punti Bézier a FREE
        k.handleTypes=(BezTriple.HandleTypes.FREE, BezTriple.
          HandleTypes.FREE)
        return k

# crea la curva blender dalla lista di liste vect
def newCurve(vect):
        """Crea una nuova curva Blender

        Data la una serie di punti definita da vect
        crea una nuova curva in Blender.

        INPUT
        -vect lista di liste di 9 float, rappresenta
              una matrice nX9 dove n è il numero di punti

        OUTPUT
        -cu   oggetto curva di blender
        """
        # controlla se l'input è valido
        if vect==None or type(vect)!=type([]) or len(vect)<=0 :
                print("ERRORE il vettore in ingresso e' vuoto")
                return Curve.New()

        # crea una nuova curva con il primo punto
        cu=Curve.New()
        cu.appendNurb(bezFromVect(vect[0]))
        cun=cu[0]
```

```
        # appende gli altri punti
        for i in vect[1:]:
                cun.append(bezFromVect(i))

        # restituisce la curva
        return cu

# aggiunge una curva ad un'altra già esistente
def addCurve(cur,vect):
        """associa curva ad un'altra esistente

        data la curva cur, associa ad essa la curva
        definita dalla lista vect.

        INPUT
        -cur    oggetto curva di blender originale
        -vect   matrice di definizione seconda curva

        OUTPUT
        -cur    oggetto curva di blender
        """

        # controllo del tipo degli input

        # controllo di cur
        if type(cur)!=type(Curve.New()):
                # se non di tipo curve ritorna nuova curva
                return NewCurve(vect)

        # controllo di vect
        if vect==None or type(vect)!=type([]) or len(vect)<=0 :
                print("ERRORE il vettore in ingresso e' vuoto")
                return cur

        # appende nuova curva a cur
        cur.appendNurb(bezFromVect(vect[0]))

        # aggiunge i punti in vect
        cun=cur[len(cur)-1]

        for i in vect[1:]:
                cun.append(bezFromVect(i))

        return cur

# chiama getCurve di matlab
def vectFromMatlab(fpath,points=20,point=None,alpha=None,beta=
  None,gamma=None,tang=None,binorm=None):
```

```
"""chiama la funzione matlab getCurve

attraverso matlabwrap chiama la funzione
getCurve di Matlab, funzione che dati in input
diversi parametri e il path di un'immagine contenente
un tubo restituisce i punti di una curva Bézier

INPUT
-fpath   stringa con il path dell'immagine da elaborare
-points  int identifica il numero di punti che si vuole
         abbia la curva
-point   opzionale tupla di 3 float del punto da cui
         si vuole abbia origine la curva
-alpha   opzionale float identifica angolo sul piano xy
         della tangente iniziale della curva
-beta    opzionale float identifica angolo sul piano xz
         della tangente iniziale della curva
-gamma   opzionale float identifica angolo sul piano yz
         della normale iniziale della curva
-tang    opzionale tupla di 3 float identifica l'asse x
         del sistema di riferimento della curva
-binorm  opzionale tupla di 3 float identifica l'asse z
         del sistema di riferimento della curva

OUTPUT
-vect    lista di liste contenente i punti della curva
"""

# verifica e cambia se è il caso la cd di matlab
if mlab.cd()!=SCRIPTPATH+r"\matlab":
        mlab._autosync_dirs = False
        mlab.cd(SCRIPTPATH+r"\matlab")

# dipendentemente dai parametri in ingresso viene
  chiamata getCurve con diversi parametri
if alpha==None or beta==None or gamma==None:
        numarr=mlab.getCurve(fpath,points)
elif tang==None or binorm==None:
        if point==None:
                numarr=mlab.getCurve(fpath,points,array
                  ((0,0,0)),alpha,beta,gamma,array
                  ((1,0,0)),array((0,0,1)))
        else:
                numarr=mlab.getCurve(fpath,points,array(
                  point),alpha,beta,gamma,array((1,0,0))
                  ,array((0,0,1)))
else:
        numarr=mlab.getCurve(fpath,points,array(point),
          alpha,beta,gamma,array(tang),array(binorm))
```

```
        # ottenuto numarr da matlab si crea la lista di liste
          vect
        vect = []
        for i in numarr:
                k = []
                for j in i:
                        k.append(j)
                vect.append(k)

        return vect
# definisce la consistenza a tubo della curva
def setTube(cur):
        """rende la curva consistente

        imposta i valori di estrusione della curva
        in modo che la curva appaia come un tubo

        INPUT
        -cur    oggetto curva da rendere tubo

        OUTPUT
        -cur    oggetto curva reso tubo
        """
        # imposta BevelDepth a 1
        cur.setExt2(1.000)
        # imposta risoluzione bev res a 5
        cur.setBevresol(5)
        # imposta i toggle della curva
        cur.setFlag(1)

        return cur

# funzione che permette il controllo di CameraBox

# chiama getPhoto di matlab
def getPhoto(direct,flash):
        """chiama la funzione matlab getPhoto

        acquisisce da remoto una fotogrrafia da
        fotocamera connessa via USB

        INPUT
        -direct   stringa con la directory dove si vuole
                  salvare l'immagine
        -flash    booleano indica se si vuole il flash

        OUTPUT
        -newpath  stringa che identifica il path
```

```
                dell' immagine salvata
        """
        # controlla e se serve imposta cd di matlab
        if mlab.cd()!=SCRIPTPATH+r"\matlab":
                mlab._autosync_dirs=False
                mlab.cd(SCRIPTPATH+r"\matlab")

        # se direct è vuoto directory di default
        if direct=="":
                direct=SCRIPTPATH+r"\\photo\\"

        # cerca file in directory
        dct=os.listdir(direct)

        # predispone nome file nuovo
        fi="image-%(num)05d.jpg"

        # se esiste già in dct aumenta contatore
        i=0
        while fi%{"num":i} in dct:
                i=i+1

        # chiama getPhoto con path definita da directory direct
          e dal nome file calcolato con fi
        newpath=mlab.getPhoto(direct+fi%{"num":i},flash)

        return newpath
```

A.2.2 MatCurve.py

```
##########################################
## MATLAB Curve: matCurve class         ##
## ---------------------------------    ##
## Questo modulo permette la gestione   ##
## della curva Matlab come oggetto.     ##
## Permette l'utilizzo di tutte le      ##
## funzioni di matCurveFunct ad         ##
## eccezione di getPhoto                ##
##                                      ##
## Author: Lorenzo "A." Mureddu         ##
##########################################

# importing di rito
from matCurveFunct import *
from Blender import Curve

# classe matCurve
class matCurve:
        """Oggetto che identifica una curva matlab
```

```
    Permette di gestire le curve ricostruite
    da un'immagine dallo script matlab
    """

    # attributo contenente l'oggetto curva blender
    _curve = None

    # costruttore
    def __init__(self,path,points=20,point=None,alpha=None,
      beta=None,gamma=None,tang=None,binorm=None):
            """costruttore, richiama getCurve

            Richiama la funzione in matCurFunct chiamata
            vectFromMatlab, passandole gli argomenti in
            input alla funzione e associa la nuova curva
            all'attributo _curve

            INPUT
            -path    stringa con il path dell'immagine da
                     elaborare
            -points  opzionale int identifica il numero di
                     punti che si vuole abbia la curva;
                     default = 20
            -point   opzionale tupla di 3 float del punto da
                     cui si vuole abbia origine la curva
            -alpha   opzionale float identifica angolo sul
                     piano xy della tang iniz della curva
            -beta    opzionale float identifica angolo sul
                     piano xz della tang iniz della curva
            -gamma   opzionale float identifica angolo sul
                     piano yz della norm iniz della curva
            -tang    opzionale tupla di 3 float identifica
                     l'asse x del sist di rif della curva
            -binorm  opzionale tupla di 3 float identifica
                     l'asse z del sist di rif della curva
            """
            self._curve = newCurve(vectFromMatlab(path,
              points,point,alpha,beta,gamma,tang,binorm))

    # modifica la curva per dare consistenza a tubo
    def makeTube(self):
            """trasforma la curva in un tubo
            """
            self._curve = setTube(self._curve)

    # getter
    def getCurve(self):
            """restituisce la curva
```

```
        OUTPUT
        -curve  curva in formato blender
        """
        return self._curve
```

A.2.3 main.py

```
##########################################################
## MATLAB Curve                                         ##
## ---------------                                      ##
## Script per blender che permette di importare         ##
## da matlab curve ricostruite partendo dalla           ##
## fotografia di un tubo.                               ##
##                                                      ##
## Author: Lorenzo "A." Mureddu                         ##
##########################################################

# importazioni moduli usati

from Blender        import Scene, Object, Draw, Window, Curve
from MatCurve       import matCurve
from matCurveFunct  import getPhoto
from math           import sqrt

# valori di default per i parametri della curva

value = 20   # numero di punti della curva
alpha = 0.0  # angolo su xy della tang iniz curva
beta  = 0.0  # angolo su xz della tang iniz curva
gamma = 0.0  # angolo su yz della norm iniz curva

# variabili globali per la gestione dei toggle dell'interfaccia,
   con relativi valori di default

pT2     = None
tube    = 1
pT3     = None
orig    = 0
flashT  = None
flash   = 0
pathB   = None
path    = ""
pToggle = None
camera  = 0

# funzioni per la gestione dell'interfaccia di blender

# gestisce la ricostruzione della curva
```

```
def addcurve(filepath):
    """permette di aggiungere una curva alla scena

    aggiunge la curva ricostruita dall'immagine in
    filepath alla scena Blender
    Viene richiamata nell'interfaccia dopo che si è
    selezionato il file dal windows file selector.

    INPUT
    -filepath rappresenta il path del file selezionato
    """

    # dichiarazione d'uso di variabili globali

    global value,alpha,beta,gamma,num,point,tube,orig

    # inizializzazione variabili locali

    block = []      # lista vuota per pup-block
    flag  = False   # identifica la selezione o meno di una curva
    curve = None    # predispone l'oggetto matCurve
    obj   = None    # predispone l'object Blender
    num   = 1       # id della curva a cui se ne associa un'altra
    point = 0       # punto sulla curva in cui attacarne un'altra

    # ottiene la scena corrente
    scn   = Scene.GetCurrent()

    # Creazione pulsanti per opzioni sulla curva

    # controlla se è selezionata o meno una curva sulla scena
    if scn.objects.active!=None and scn.objects.active.getType()
      =="Curve":
        # crea number sul punto della curva
        p     = Draw.Create(point)
        # oggetto selezionato ergo flag a true
        flag  = True
        # ottiene l'oggetto curva attivo
        obj   = scn.objects.active
        # ottiene la curva dall'oggetto
        curve = obj.getData()

        # controlla se la curva è strutturata
        if len(curve)>1:
            # seleziona con pup la curva da utilizzare
            num = Draw.PupIntInput("selez curva:", num, 1, len(
              curve))

    # crea number per numero di punti
```

```
v = Draw.Create(value)

if not orig:
    # crea number per angoli alpha beta e gamma
    a = Draw.Create(alpha)
    b = Draw.Create(beta)
    c = Draw.Create(gamma)

# appende i number nel blocco
block.append(("numero di punti:",v, 5, 500))

if flag:
    block.append(("punto in cui inserire:",p, 0,len(curve[
      num-1])-1))

if not orig:
    block.append(("alpha:",a, 0.0, 360.0))
    block.append(("beta:",b,0.0,360.0))
    block.append(("gamma:",c,0.0,360.0))

# disegna blocco
Draw.PupBlock("Parameters of the curve",block)

# assegna valori a variabili globali
value = v.val

if not orig:
    alpha = a.val
    beta  = b.val
    gamma = c.val

# disegna curva dipendentemente da flag

# se la curva è aggiunta a una precedente
if flag:
    point = p.val
    # ottiene curva def da num
    curr  = curve[num-1]
    # ottiene BezTriple def da value
    ptemp = curr.__getitem__(point).getTriple()
    # ottiene il punto
    knot  = (ptemp[1][0],ptemp[1][1],ptemp[1][2])

    # se il punto def da value non è l'ultimo
    if point !=len(curr)-1:
        # calcolo tangente e derivate 1 e 2 del punto
        tang  = (ptemp[2][0]-ptemp[1][0],ptemp[2][1]-ptemp
          [1][1],ptemp[2][2]-ptemp[1][2])
        p1    = curr.__getitem__(point+1).getTriple()
```

```
            xder   = [3*ptemp[2][0]-3*ptemp[1][0],3*ptemp
              [2][1]-3*ptemp[1][1],3*ptemp[2][2]-3*ptemp[1][2]]
            xder2 = [6*p1[0][0]-12*ptemp[2][0]+6*ptemp[1][0],6*
              p1[0][1]-12*ptemp[2][1]+6*ptemp[1][1],6*p1
              [0][2]-12*ptemp[2][2]+6*ptemp[1][2]]

        # se il punto def da value è l'ultimo
        else:
            # calcolo tangente e derivate 1 e 2 del punto
            tang   = (ptemp[1][0]-ptemp[0][0],ptemp[1][1]-ptemp
              [0][1],ptemp[1][2]-ptemp[0][2])
            p0     = curr.__getitem__(point-1).getTriple()
            xder   = [3*ptemp[1][0]-3*ptemp[0][0],3*ptemp
              [1][1]-3*ptemp[0][1],3*ptemp[1][2]-3*ptemp[0][2]]
            xder2 = [6*ptemp[1][0]-12*ptemp[0][0]+6*p0[2][0],6*
              ptemp[1][1]-12*ptemp[0][1]+6*p0[2][1],6*ptemp
              [1][2]-12*ptemp[0][2]+6*p0[2][2]]

        # calcolo binormale
        binorm = (xder[1]*xder2[2]-xder[2]*xder2[1],xder[2]*
          xder2[0]-xder[0]*xder2[2],xder[0]*xder2[1]-xder[1]*
          xder2[0])
        # crea curva
        mcur    = matCurve(filepath,value,knot,alpha,beta,gamma,
          tang,binorm)
        # appende la curva alla scena corrente
        obj2    = scn.objects.new(mcur.getCurve())
        # modifica la nuova curva come quella originale
        obj2.setLocation(obj.LocX,obj.LocY,obj.LocZ)
        obj2.RotX,obj2.RotY,obj2.RotZ=obj.RotX,obj.RotY,obj.RotZ
        obj2.setSize(obj.SizeX,obj.SizeY,obj.SizeZ)
        # associa la nuova curva a quella originale
        obj.join([obj2])
        # cancella la nuova curva
        scn.objects.unlink(obj2)

    # se la curva è la prima
    else:

        # crea la curva dip da toggle orig
        if orig:
            curve = matCurve(filepath,value)
        else:
            curve = matCurve(filepath,value,None,alpha,beta,
              gamma)

        # dip da toggle tube da cons alla curva
        if tube:
            curve.makeTube()
```

```
        # crea curva nella scena
        obj=scn.objects.new(curve.getCurve())

        # trasla la curva nella posizione del cursore
        pos=Window.GetCursorPos()
        obj.setLocation(pos[0],pos[1],pos[2])

def button(evt): # gestisce gli eventi dei pulsanti
    """gestisce gli eventi dei pulsanti nell'interfaccia di
      blender.

    INPUT
    -evt identifica l'evento occorso
    """

    # dichiarazione d'uso variabili globali

    global pToggle,pathB,path,flashT,pT2,camera,flash,tube,orig

    # gestisce eventi dei button, dip da evt

    if evt==1:    # button curve

        if not camera: # toggle camera off
            # attiva selettore di file e chiama addcurve
            Window.FileSelector(addcurve,"Selezionare immagine:"
              )
        else:          # toggle camera on
            # chiama getPhoto
            st=getPhoto(path,flash)

            if st!="error": # se scatto avvenuto
                addcurve(st) # si chiama addcurve con filepath
                  da getPhoto
    elif evt==2:   # button quit
        Draw.Exit() # termina

    # cambia valore ai toggle
    elif evt==3:   # toggle camera
        camera=pToggle.val
    elif evt==5:   # toggle tube
        tube=pT2.val
    elif evt==7:   # toggle orig
        orig=pT3.val
    elif evt==6:   # toggle flash
        flash=flashT.val

    # imposta path dove salvare fotografie
```

```
    elif evt==4:   # String pd
        try:    # controlla se filepath valido
            import os
            os.listdir(pathB.val)
            del os
            # aggiorna il path
            path=pathB.val
        except: # se non valido non cambia niente
            pathB.val=path
            Draw.Redraw(1)

def gui(): # gestisce l'interfaccia grafica
    """gestisce l'interfaccia grafica di blender e richiama gli
      eventi
    """

    # dichiarazione d'uso variabili globali

    global pToggle,pathB,pT2,flashT,pT3,path,camera,tube,orig,
      flash

    # disegna i vari pulsanti dell'interfaccia grafica

    # i 2 button
    Draw.PushButton("Curve",1,10,100,200,20,"draw matlab Curve")
    Draw.PushButton("quit",2,10,25,200,20,"exit")

    # i 4 toggle
    pToggle = Draw.Toggle("Photo", 3, 10, 50, 50, 20, camera, "
      select if get the photo from camera or from file")
    pT2     = Draw.Toggle("Tube", 5, 110, 50, 50, 20, tube, "
      select if you want tube or curve")
    pT3     = Draw.Toggle("Orig", 7, 160, 50, 50, 20, orig, "
      select if you want original position")
    flashT  = Draw.Toggle("Flash", 6, 60, 50, 50, 20, flash, "
      select if photo whit or whitout flash")

    # lo string
    pathB   = Draw.String("pd", 4, 10, 75, 200, 20, path, 50,"
      select where store the photo if none store in scriptpath")

# Main

# unica funzione eseguita dallo script registra l'interfaccia in
    blender
Draw.Register(gui,None,button)
```

A.2.4 setup.py

```
##########################################
## MATLAB Curve: setup                   ##
## ----------------------                ##
## Script per impostare correttamente    ##
## il path della cartella contenente     ##
## i vari script di MATLAB curve         ##
## nel modulo matCurveFunct              ##
##                                       ##
## Author:Lorenzo "A." Mureddu           ##
##########################################

# importa il modulo di sistema operativo
import os

# ottiene la directory corrente
PATH    = os.getcwd()
# apre il file matCurveFunct
fi      = open(PATH+r"\matCurveFunct.py",'r')
# legge tutte le linee
i       = fi.readlines()
# crea flag identifica diversità nella path
changed = False
# chiude il file
fi.close()

# cerca la linea contenente SCRIPTPATH= nel file
for k in i:
        if k.find("SCRIPTPATH=")!=-1:
                # controlla se coincide con la cd
                if k != "SCRIPTPATH=r'"+PATH+"'␣␣␣\n":
                        # cambia la riga in matCurveFunct
                        i.insert(i.index(k),"SCRIPTPATH=r'"+PATH
                          +"'␣␣␣\n")
                        i.remove(k)
                        chenged=True
                        print("updating␣file...whit␣"+"
                          SCRIPTPATH=r'"+PATH+"'␣␣␣\n")
                break

# se è stata cambiata la riga sovrascrive il file matCurveFunct
if changed:
        fi=open(PATH+r"\matCurveFunct.py",'w')
        for j in i:
                fi.write(j)
        fi.close()
```

Appendice B

Il manuale utente

In questa appendice è contenuto un'abbozzo di manuale utente del progetto realizzato, chiamato MATLAB Curve. Non viene descritta nello specifico l'interfaccia utente, già descritta precedentemente nella sezione 4.5, ma solo come installare e avviare il software.

B.1 Installazione

Per poter utilizzare il software MATLAB Curve è necessario che siano installati nel computer i seguenti software almeno nelle versioni citate o superiori:

- Matlab 2006
- Blender v 2.45
- Python v 2.5
- MatlabWrap v 1.0
- libreria Numpy per Python
- libreria Canon SDK

A questo punto bisogna copiare la cartella `matlabCurve` in una cartella sul proprio hard disk. Successivamente si avvia il programma python al suo interno chiamato `setup.py` e si imposta la variabile d'ambiente `PYTHONPATH` uguale alla directory in cui è stata copiata la cartella `matlabCurve`, comprensiva della cartella stessa, alle directory in cui sono installate le librerie python stesse e alla directory di MatlabWrap.

B.2 Utilizzo

Per utilizzare MATLAB Curve si avvia Blender, si carica e si avvia lo script python chiamato `main.py` nella cartella `matlabCurve`. A questo punto comaparirà a schermo l'interfaccia utente.

Per ottenere la curva bisogna premere il pulsante **Curve** sull'interfaccia grafica. Se il toggle **Photo** è selezionato, e la fotocamera è correttamente collegata e accesa, viene scattata la fotografia, altrimenti viene mostrata una schermata per selezionare un file dell'immagine da cui ricostruire la curva. Se si preme il pulsante **Curve** e nella scena corrente di Blender è selezionata una curva, la nuova curva creata viene innestata sulla curva selezionata, altrimenti la curva viene creata ex-novo con i parametri che gli vengono forniti.

Per tutto il resto si veda la sezione 4.5.

Appendice C

Esempi di impiego

Questa appendice contiene esempi di impiego del software nel rendering di alcune immagini e animazioni.

Tutti questi lavori sono stati ottenuti facilmente grazie alle possibilità di sviluppo offerte da Blender.

C.1 Immagini

La prima immagine, figura C.1, rappresenta il primo tentativo di costruzione di una struttura complessa a più curve. La curva ivi rappresentata è ottenuta innestando due curve nello stesso punto con angoli rispettivamente di $\alpha = 90°$ e $\beta = 0°$ nella prima curva, e $\alpha = 90°$ e $\beta = 180°$ nella seconda curva. Nota che si è omesso γ poichè in una prima fase di sviluppo, la rotazione della curva veniva identificata dai soli angoli α e β.

Nella seconda immagine, figura C.2, viene visualizzata una struttura complessa composta da tre curve, di cui due innestate nello stesso punto con valori uguali a $0°$ per α e β e valori diversi di γ. Questo rendering è nato come testing per il controllo che l'angolo γ imponeva alla curva.

La terza immagine, figura C.3, viene riportata come esempio per la creazione di un logo: il simbolo al centro, infatti, è stato costruito usando una curva chiusa bidimensionale, che definisce, quindi, una superfice. Sebbene non derivante da un'immagine di un tubo, la curva chiusa definente il simbolo è stata creata durante una precisa fase del testing; nell'iniziale processo di comunicazione tra Matlab e Blender veniva usato un file, per testare questo processo inizialmente si sono inserite manualmente le coordinate di una curva 2d prestabilita nel file da cui sarebbe stata disegnata la stessa. Quella visualizzata in figura C.3 è proprio la curva bidimensionale utilizzata.

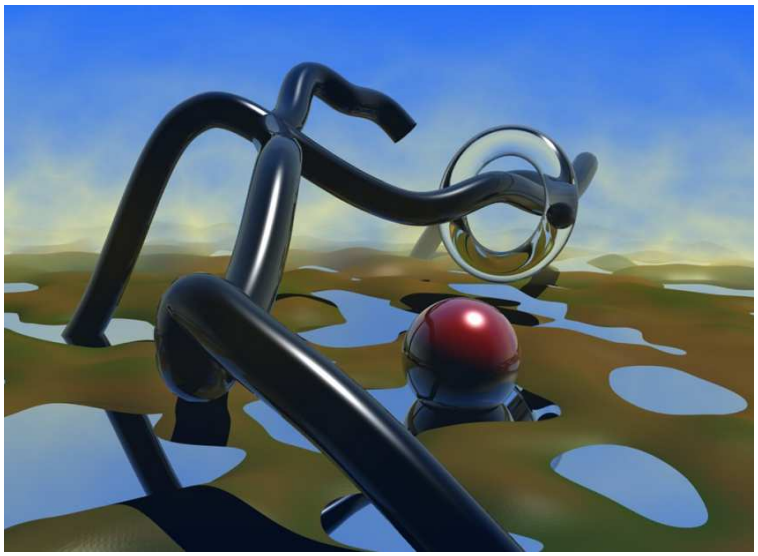


Figura C.1: prima immagine: struttura con due rami ortogonali

Figura C.2: seconda immagine: struttura complessa multi-curve

L'ultima immagine, figura C.4, rappresenta un'esempio di oggetto complesso deformato da una curva. In quest'ultima immagine è visibile la testa di scimmia rossa deformata secondo la curva definita da una linea nera; per evidenziare la differenza con l'oggetto normale è stata inserita nella scena la stessa testa di scimmia di colore diverso, blu.

Figura C.3: terza immagine: curva 2d per creazione di un logo

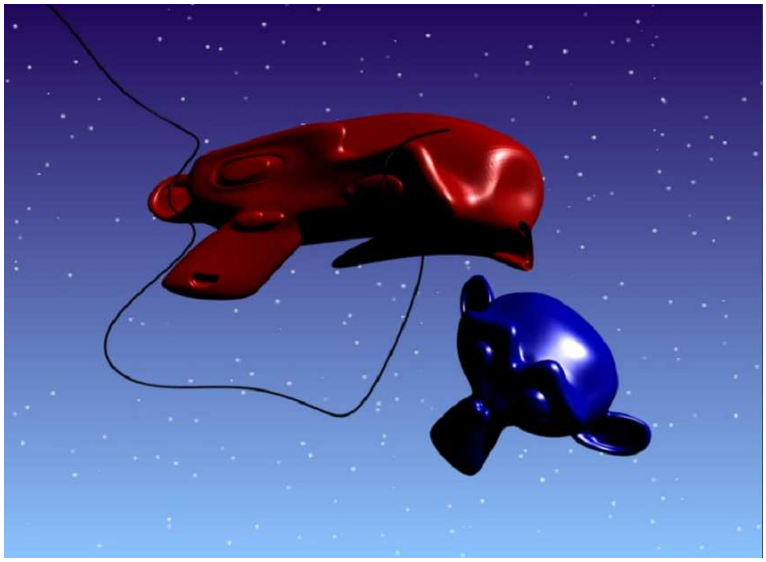


Figura C.4: quarta immagine: deformazione di un oggetto tramite curva

C.2 Animazioni

Le animazioni qui presentate sono state sviluppate come esempi dell'inseguimento della curva da parte di oggetti.

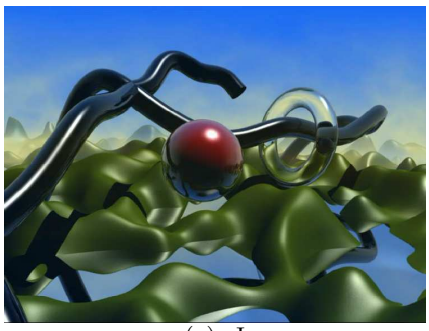
Il primo video, figure C.5, rappresenta un primo tentativo di rendering di un'animazione, riprende l'immagine C.1, modificandola leggermente, e rappresenta lo stesso tipo di curva dell'immagine sopra citata. La telecamera si muove in modo lineare da un punto all'altro senza seguire alcuna curva. Da notare il dettaglio raggiungibile sulla superficie della curva nel settimo frame presentato, definito da alti valori di Bevel resolution e di risoluzione della curva.

Il secondo video, immagini C.6 rappresenta un esempio di struttura a più curve complessa. Nell'intenzione dell'autore la curva rappresenta la possibilità di sviluppare successivamente un'algoritmo per la creazione di alberi a complessità variabile. Le curve sono innestate una sull'altra consecutivamente, la seconda appoggia sulla prima, la terza sulla seconda e così via. La telecamera, infine, si muove su una traiettoria circolare attorno alla struttura multicurve visualizzata.

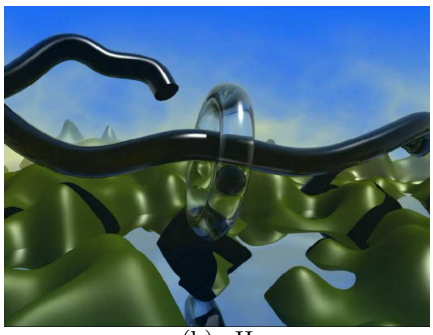
Il terzo video, immagini C.7, rappresenta una sfera rossa che si muove all'interno di un tubo trasparente. Questa sfera rossa deve il suo movimento alla curva definita dal tubo, di cui ne insegue la traccia. La telecamera in questo frangente si muove linearmente tra punti predefiniti, e segue la curva mantenendola sempre al centro della scena.

Il quarto video, immagine C.8, riprende il video C.7, cambiando però il punto di vista della telecamera. Quest'ultima punta sempre alla sfera rossa, come nel caso precedente, ma in questo nuovo video si muove all'interno del tubo con essa, seguendo essa stessa la traiettoria definita dal tubo trasparente. Questo esempio è stato creato per mostrare come qualsiasi cosa possa seguire la traiettoria definita da una curva. Il video raffigurato in C.8 introduce un possibile sviluppo futuro del software, la creazione da parte dell'utente stesso di montagne russe virtuali, che associate a un simulatore di movimenti, sono tanto in voga nei luna park.

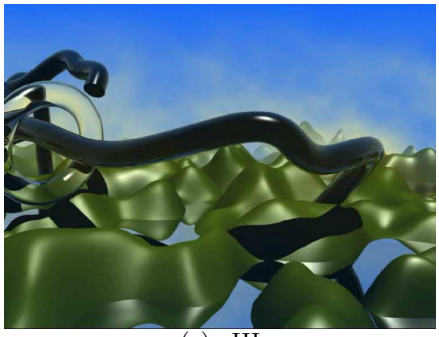
Il quinto, e ultimo, video, immagini C.9, mostra un aeroplanino che segue una traiettoria definita da una curva invisibile. La curva invisibile in questione è quella a forma di spirale ottenuta nelle immagini 6.9. Questo video mostra quindi un esempio in cui la curva, sebbene non appaia esplicitamente, influisce in modo decisivo sul moto degli oggetti nella scena.



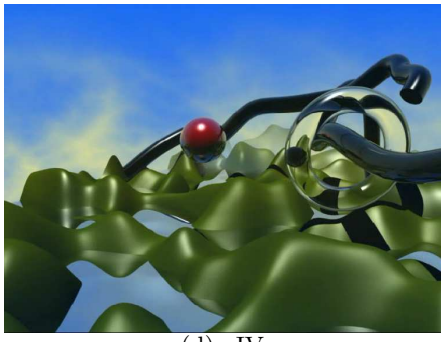
(a) I



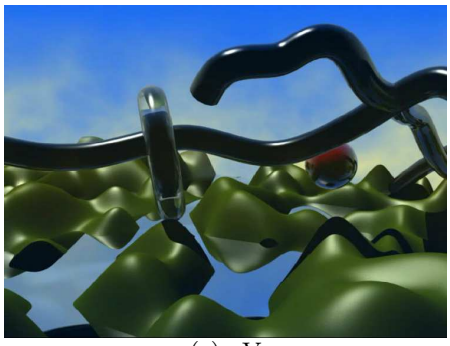
(b) II



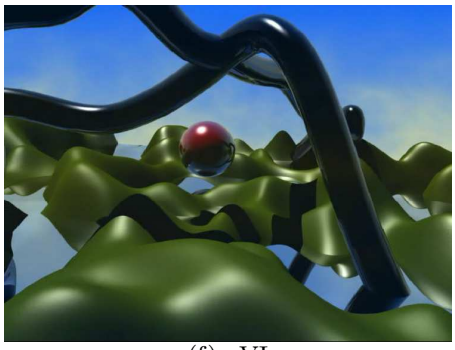
(c) III



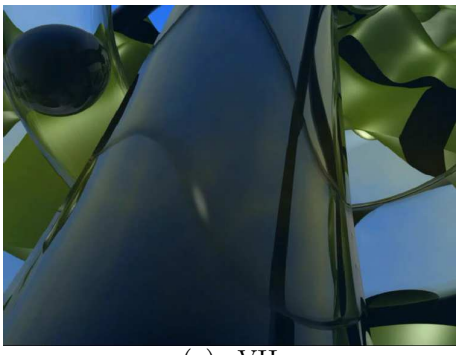
(d) IV



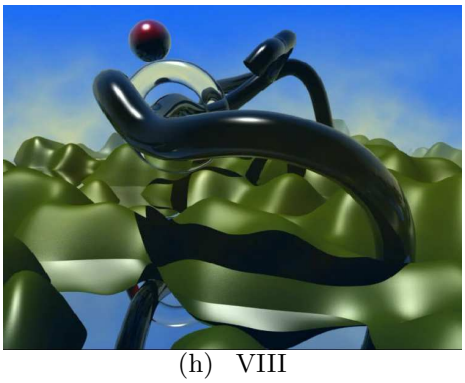
(e) V



(f) VI

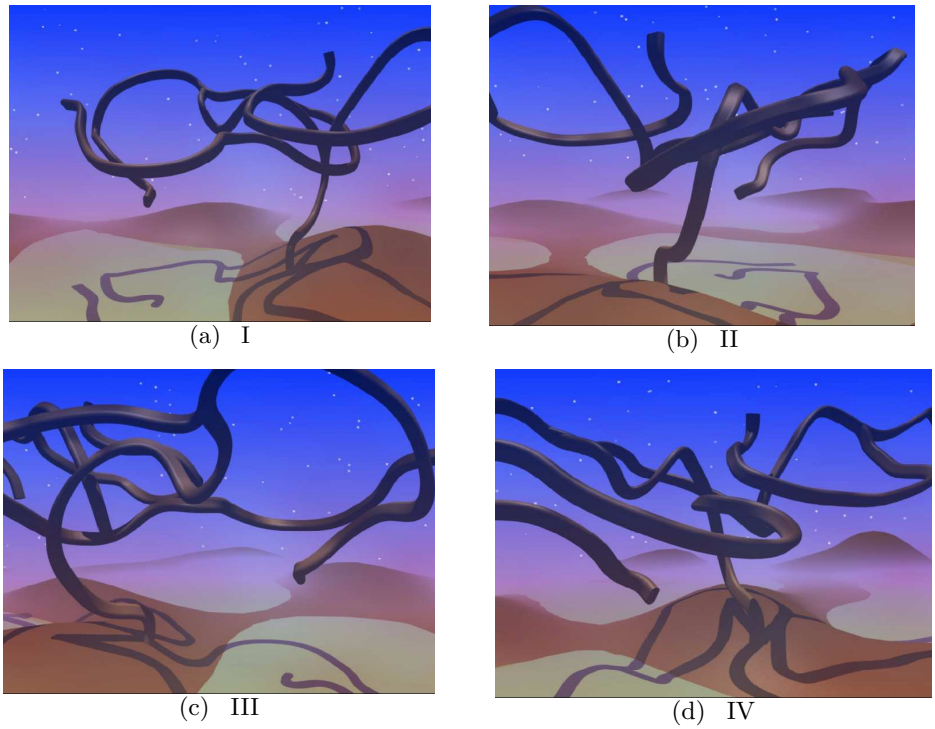


(g) VII



(h) VIII

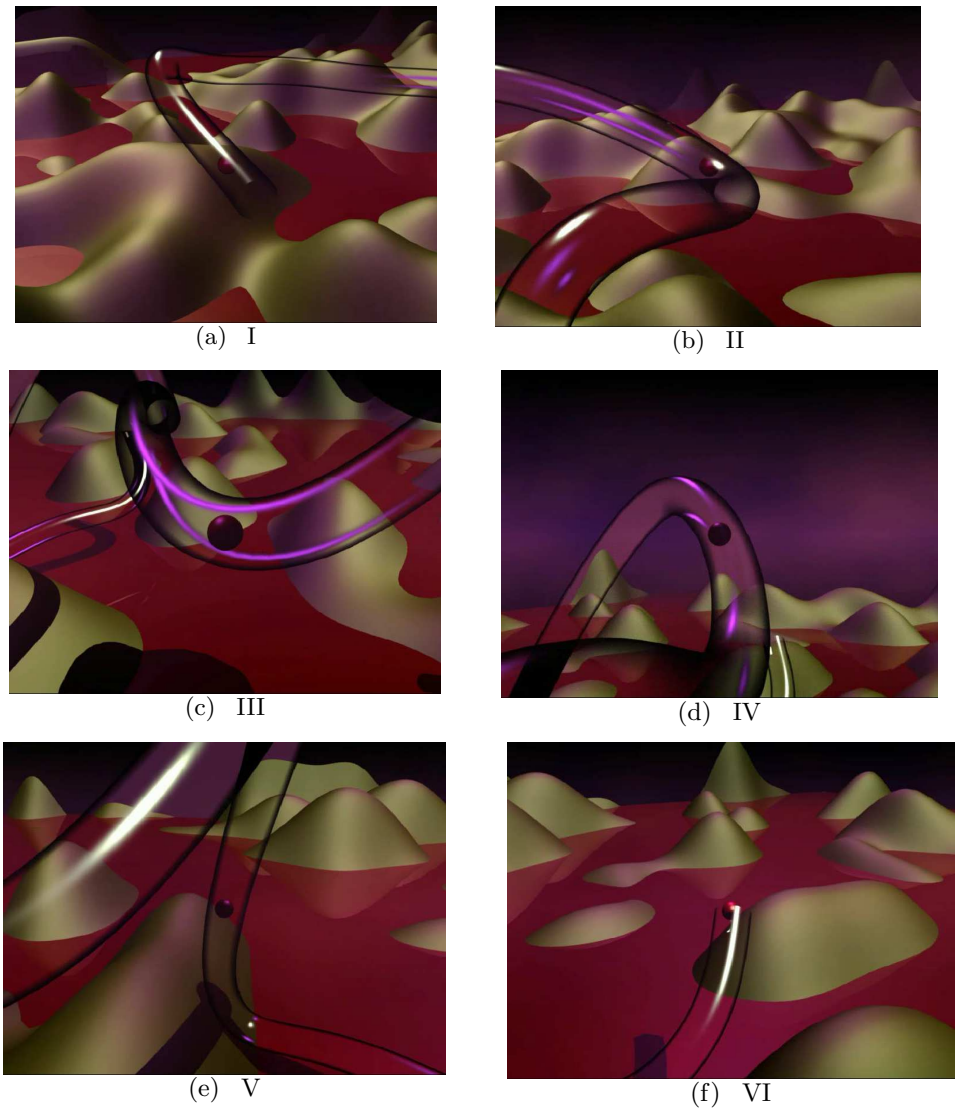
Figura C.5: primo video: struttura di complessità media



(a) I (b) II

(c) III (d) IV

Figura C.6: secondo video: struttura complessa

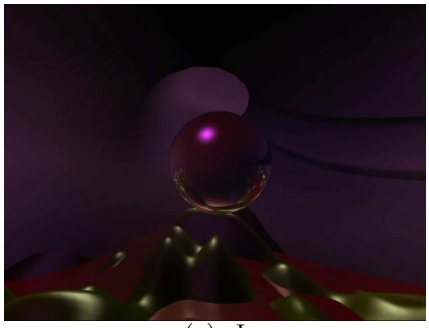


(a) I (b) II

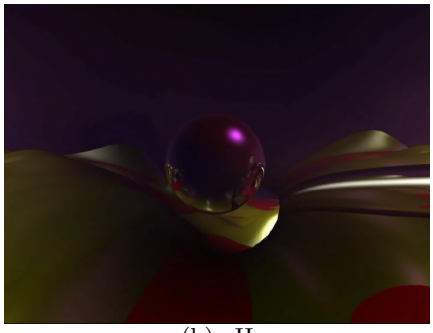
(c) III (d) IV

(e) V (f) VI

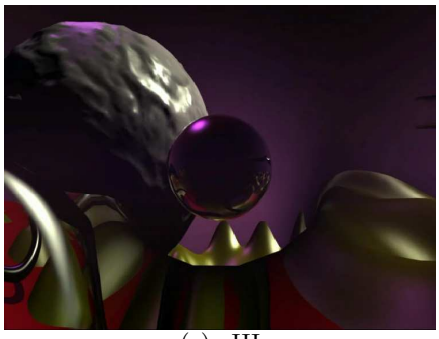
Figura C.7: terzo video: inseguimento di curva da parte di un oggetto



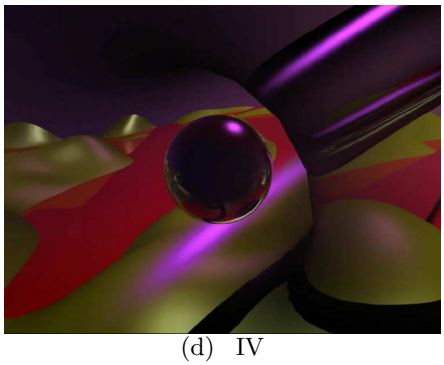
(a) I



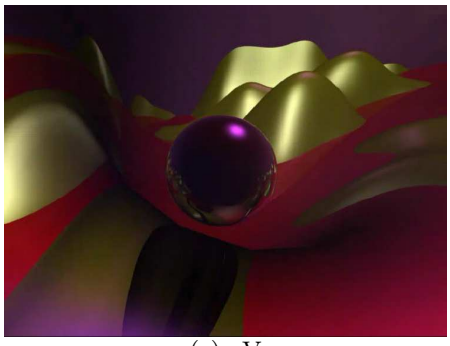
(b) II



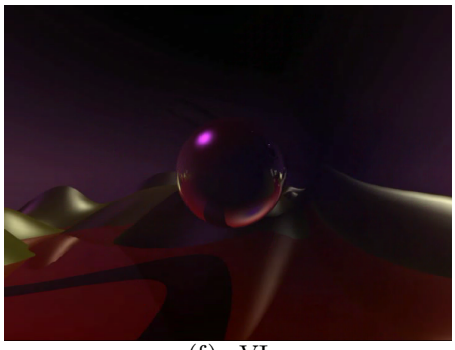
(c) III



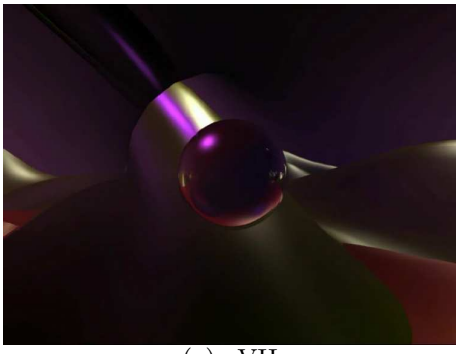
(d) IV



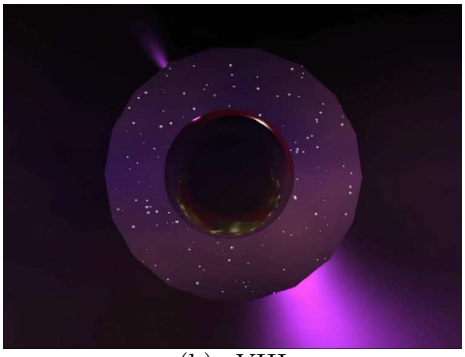
(e) V



(f) VI



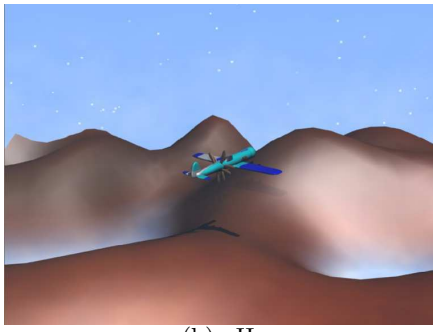
(g) VII



(h) VIII

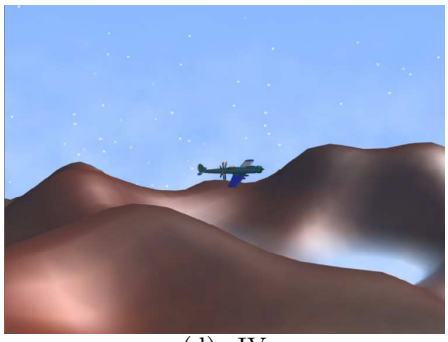
Figura C.8: quarto video: oggetto e telecamera inseguono curva

(a) I



(b) II

(c) III



(d) IV

(e) V

(f) VI

(g) VII

(h) VIII

Figura C.9: quinto video: oggetto insegue curva invisibile