UNIVERSITÀ DEGLI STUDI DI PARMA
FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica

# SVILUPPO DI UN SISTEMA PER LA SIMULAZIONE GRAFICA DI FOLLE IN MOVIMENTO

Relatore:
Chiar.mo Prof. Ing. STEFANO CASELLI

Correlatori:
Ing. JACOPO ALEOTTI

Tesi di laurea di:
RITA BELTRAMI

26 gennaio 2010

*A mio zio,*
*Marco Beltrami*

*Il primo e più grande ringraziamento va al Dott. Ing. Jacopo Aleotti, che è sempre stato estremamente disponibile e gentile con me, aiutandomi con grande umanità ogni volta che avevo un dubbio o un problema.*

*Ringrazio inoltre il mio relatore, il Prof. Caselli, che mi ha seguita con estrema cortesia e preziosi consigli.*

*Ringrazio poi i miei amici dell'università, sia quelli che hanno condiviso con me ogni giorno di lezione nel corso della laurea triennale (Monica, Isabella, Francesca, Francesco, Alan, Rossano e Alessandro), sia quelli che mi hanno fatto ridere e tenuto compagnia nelle giornate di lavoro in laboratorio (soprattutto Marco, Luca e Christian). E perché no, un saluto anche al robot Comau, che scattando all'improvviso dietro alle mie spalle mi faceva sempre spaventare!*

*Infine, un ringraziamento alla mia famiglia e a Federico, che ha cercato di tirarmi su di morale quando mi preoccupavo per essere rimasta indietro.*

“È molto più bello sapere qualcosa di tutto, che tutto di una cosa.”

*Blaise Pascal*

# Indice


**1 Introduzione** **1**

**2 La simulazione di folle** **3**

2.1 Introduzione . . . . . 3

2.2 Stato dell'arte . . . . . 4

2.3 Creazione dei modelli umani . . . . . 7

2.3.1 Sintesi dei modelli . . . . . 7

2.3.2 Differenziare i modelli . . . . . 8

2.4 Animazione dei modelli . . . . . 9

2.5 Animazione comportamentale della folla . . . . . 11

2.6 Pianificazione delle traiettorie . . . . . 13

2.6.1 Soluzioni proposte nel mondo della robotica . . . . . 13

2.6.2 Pianificazione di traiettorie per folle . . . . . 14

2.7 Correlazione tra folle virtuali e folle reali . . . . . 17

2.8 Level Of Detail . . . . . 19

**3 Strumenti software utilizzati** **23**

3.1 Horde3D . . . . . 23

3.2 ColladaConverter . . . . . 25

3.2.1 Il formato COLLADA . . . . . 26

3.3 Google SketchUp Pro 7 . . . . . 29

3.4 Autodesk 3ds Max 9 . . . . . 30

3.5 CubeMapGen . . . . . 31

**4 Descrizione del sistema** **35**
4.1 Descrizione generale . . . . . 35
4.2 Sviluppo del sistema . . . . . 37
4.3 Animazione dei personaggi . . . . . 39
4.4 Struttura del codice . . . . . 44
4.5 Creazione dello scenario . . . . . 46
4.6 Movimento dei gruppi . . . . . 49
4.6.1 Schieramento iniziale . . . . . 49
4.6.2 Scelta della destinazione futura . . . . . 50
4.6.3 Collision avoidance . . . . . 53
4.6.4 Collision avoidance con membri di altri gruppi . . . . . 58
4.7 Analisi delle prestazioni . . . . . 61

**5 Conclusioni** **63**

**Bibliografia** **64**

# Capitolo 1

# Introduzione

La simulazione grafica di folle in movimento è un ambito di studio in continuo sviluppo, che vede impegnati numerosi gruppi di ricerca in tutto il mondo. Le sue applicazioni riguardano molteplici campi, che vanno dall'industria dell'intrattenimento (videogiochi, industria cinematografica), all'addestramento di forze di polizia e militari (simulazione di manifestazioni e rivolte), all'architettura (progettazione di città ed edifici), alla scienza della sicurezza (evacuazione di edifici, navi, piazze), alla sociologia (studio dei comportamenti della folla). Ovviamente, a diverse applicazioni corrisponderanno differenti approcci.

Ma che cos'è una folla? Potremmo definirla come "un vasto gruppo di individui nello stesso ambiente fisico che condividono un obbiettivo comune e potrebbero agire in modo diverso rispetto a quanto farebbero se fossero da soli". Simulare una folla1.1 significa riprodurla in tutti i suoi aspetti in un ambiente non più reale ma virtuale. Si tratta di un problema complesso, che richiede di prendere in considerazione varie tematiche. Innanzitutto bisogna creare modelli virtuali delle singole persone e far sì che essi abbiano tutti un aspetto diverso. Il passo successivo è fare in modo che anche il loro comportamento risulti diversificato. La folla nel suo complesso dovrà poi seguire comportamenti realistici anche a livello di gruppo e navigare nell'ambiente virtuale secondo regole precise. Infine, bisogna occuparsi di aspetti di rendering, allo scopo di ottenere un risultato complessivo visivamente gradevole pur mantenendo i costi e i tempi di calcolo il più possibile ridotti.

Nell'ambito di questa tesi, si è sviluppato un simulatore grafico di folle appog-

**Figura 1.1:** Esempio di folla virtuale.

giandosi sul motore grafico open source Horde3D, sviluppato dall'università di Asburgo. Oltre a questo, vedremo che sono stati utilizzati molti altri software, ognuno necessario ad affrontare un aspetto del problema.

Tra le varie tematiche che, si è detto, devono essere affrontate nel simulare una folla, ci si è in particolare concentrati sull'animazione dei singoli modelli umani ma, soprattutto, su come far seguire alle persone determinate traiettore mantenendo una loro identità e compattezza a livello di gruppo ed evitando le collisioni tra un pedone e l'altro. Il codice del simulatore è stato scritto in linguaggio C++, utilizzando ove necessario alcune specifiche funzioni per la gestione di risorse grafiche esterne messe a disposizione da Horde3D.

Nelle pagine seguenti, dopo una panoramica generale sul mondo della simulazione di folle, verranno dunque presentati i principali strumenti software utilizzati, per poi passare ad una descrizione più puntuale del progetto sviluppato. Verranno illustrate le fasi del lavoro svolto, con particolare attenzione ai passi necessari per creare un modello animato e agli algoritmi realizzati per garantire il movimento dei gruppi di persone, per poi concludere con un'analisi delle prestazioni del simulatore sviluppato e alcune indicazioni su eventuali possibili miglioramenti ed utilizzi futuri dello stesso.

# Capitolo 2

# La simulazione di folle

## 2.1 Introduzione

Sebbene il comportamento collettivo sia stato oggetto di studio sin dalla fine del diciannovesimo secolo, i primi tentativi di simularlo al calcolatore risalgono alla metà degli anni Novanta del ventesimo secolo. É possibile distinguere due grandi aree nella simulazione di folle. La prima si concentra su aspetti di tipo comportamentale, trascurando la qualità del risultato visivo; nella maggior parte dei casi i membri della folla sono rappresentati in modo schematico, tipicamente come puntini colorati o figure stilizzate, mentre l'attenzione si concentra sul realismo delle azioni e dei movimenti delle persone. Nella seconda area, l'obbiettivo principale è l'alta qualità della visualizzazione (per esempio nell'industria cinematografica e nei videogame), mentre solitamente il realismo nei comportamenti non è prioritario. Ovviamente ci sono anche studi che puntano a conciliare le due opposte esigenze.

Come suggerisce l'esperienza di tutti i giorni, gli umani che compongono una folla virtuale devono avere un aspetto tra loro diverso, seguire traiettorie diverse, compiere azioni diverse e così via. La sfida principale diventa pertanto quella di ottenere una simulazione il più realistica possibile pur mantenendo la richiesta di risorse computazionali limitata.

## 2.2 Stato dell'arte

Vediamo di seguito alcuni dei principali settori nei quali si è tentato di modellare il comportamento delle folle.

Una dei campi in cui il comportamento delle folle è stato maggiormente studiato è quella degli studi di sicurezza (figura 2.1), al fine di realizzare simulatori di evacuazioni di emergenza. Lo scopo è quello di capire se sia possibile evacuare un dato edificio in tempi ragionevoli, identificare la presenza di ostacoli pericolosi e capire se e dove si creano zone in cui la pressione esercitata dalla folla di persone sia eccessiva. Uno dei software più utilizzati in questo senso è *Legion* [1], impiegato, tra le altre cose, anche per gli studi di sicurezza precendenti alle olimpiadi di Sydney 2000.

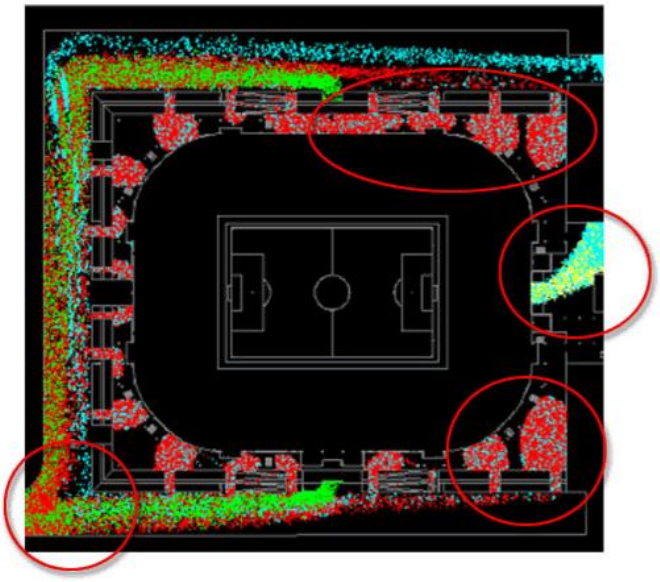

**Figura 2.1:** Legion utilizzato per lo studio della sicurezza e dei tempi di uscita dallo stadio Düsseldorf Arena. I cerchi rossi indicano le aree a maggiore congestione.

Un certo interesse in questo ambito è stato dimostrato anche dagli studi architettonici. Un software in grado di simulare graficamente folle in movimento potrebbe infatti essere utilizzato per simulare visitatori virtuali degli edifici o anche degli spazi aperti che si vanno a progettare, in modo da offrire al cliente un'immag-

ine visiva di quale sarà l'aspetto assunto dalla costruzione al termine della sua realizzazione.

La modellazione di folle è stata essenziale anche nei sistemi di simulazione usati per addestrare i militari o le forze di polizia al fine di studiare grandi assemblamenti di persone. Per esempio *Small Unit Leader Non-Lethal Training System*[2] è un simulatore in dotazione ai marines degli Stati Uniti per prepararli ad operazioni di peacekeeping e controllo di folle, al fine di addestrare i soldati a prendere decisioni in situazioni in cui è consentito il solo uso di munizioni non letali.

Sebbene si tratti di un settore in cui è di primaria importanza lo studio del comportamento collettivo, al contrario di quanto ci si aspetterebbe ci sono in realtà solo pochi progetti di simulazione di folle condotti nell'ambito della sociologia.

Importanti contributi provengono invece dal mondo della robotica e della ricerca sull'intelligenza artificiale. I ricercatori sono interessati a studiare come comportamenti di gruppo emergano da regole locali. La principale fonte di ispirazione è la natura, dove ad esempio insetti quali le api o le formiche risolvono problemi come trovare il cibo, costruire nidi, o dividersi il lavoro solamente interagendo individualmente, senza nessun supervisore a livello globale. Questi studi hanno avuto diverse applicazioni, dalla creazione algoritmi per la soluzione di problemi di routing [3] alla progettazione sistemi per il controllo del movimento di flotte di robot autonomi [4].

Uno dei settori nei quali negli ultimi anni si è registrata una notevole velocità di sviluppo nel campo della simulazione di folle è quello degli effetti speciali. Se fino a una quindicina di anni fa non esisteva alcun modo per ricreare folle digitali, oggi quasi tutti i film di maggior successo ne contengono alcune, tendenza che ora anche video musicali, telefilm e pubblicità iniziano a seguire. L'utilizzo di folle virtuali non solo abbassa notevolmente i costi di produzione, ma consente anche una maggior libertà creativa, grazie alla loro flessibilità. Spesso, inoltre, si usano tecniche miste, per cui i personaggi in primo piano sono attori veri, mentre quelli sullo sfondo sono persone simulate. Ovviamente, in questo caso l'obbiettivo principale è quello di ottenere una resa visiva il più possibile realistica, esigenza a cui consegue un rilevante costo computazionale: tuttavia in questo caso i tempi di elaborazione non sono un fattore così penalizzante, perché si sta parlando di simulazioni che non sono certamente di tipo real-time. Per girare pellicole con alle spalle grossi budget

si tende solitamente a creare strumenti *ad hoc* che verranno utilizzati solo per quel particolare film. Esistono però anche applicativi più versatili che sono stati adottati in più pellicole; tra questi, il più famoso software per produzioni non real-time (cinematografiche e non solo) è sicuramente *Massive* [5], impiegato ad esempio per produzione di film quali *Il Signore degli Anelli*, *I Robot* (figura 2.2), *Avatar* e molti altri.

**Figura 2.2:** Scena di *I Robot*, film realizzato con Massive

Le situazione è più complessa all'interno di un videogioco, perché qui la velocità di calcolo deve unirsi ad un rendering soddisfacente. Per questo motivo, spesso si usa una tecnica denominata **level-of-detail** (LOD): essa prevede di calcolare il comportamento solo dei personaggi visibili o che risulteranno visibili di lì a poco. Gli individui vengono creati solo nello spazio circostante il giocatore, che vive dunque in una sorta di bolla di simulazione. Tutto il resto della folla non è altro che una texture o l'insieme porzioni di varie texture accostate.

Infine, la simulazione di folle gioca un ruolo chiave anche nei progetti di realtà virtuale. Il principale obbiettivo diventa immergere lo spettatore in uno spazio virtuale popolato da umani digitali e consentirgli di esplorare la scena in modo che ciò che egli osserva sembri il più possibile realistico.

## 2.3 Creazione dei modelli umani

### 2.3.1 Sintesi dei modelli

Nel simulare una folla persone la prima cosa da fare è sintentizzare dei modelli di persone umane (figura 2.3). Una possibile tecnica per ottenerli è quella di produrli manualmente. Un primo modo di procedere, simile a quello seguito da uno scultore, è quello di partire da una forma geometrica di base, quale ad esempio un parallelepipedo, e poi deformarla fino ad ottenere sembianze umane. Un'altra strada è invece quella di creare un modello "multistrato" (o "muscoloscheletale"), ossia cominciare costruendo una sorta di scheletro e cui si andranno via via ad aggiungere livelli intermedi per simulare il volume del corpo (muscoli, grasso, ecc), rivestendo infine tutto con uno strato di pelle.

**Figura 2.3:** Esempio di template di donna.

Una procedura più semplice e veloce consente di ricostruire al calcolatore la struttura goemetrica di una persona reale attraverso l'utilizzo di scanner 3D[1] oppure

[1]Uno scanner 3D è un dispositivo che analizza oggetti reali per raccogliere informazioni sulla loro forma (e possibilmente anche sul colore). I dati ottenuti possono poi essere utilizzati per costruire modelli digitali tridimensionali.

di video e fotografie della persona stessa. Questo procedimento è efficiente se si mira ad ottenere umani virtuali con un aspetto realistico, ma se invece si vogliono realizzare personaggi più di fantasia le cose si complicano, perché ottenere il modello ottenuto non è così facile.

### 2.3.2 Differenziare i modelli

Suppondendo di avere una folla costituita da un numero considerevole di persone, risulta improponibile pensare di realizzare manualmente ogni singolo modello: si dovrebbero affrontare problemi insormontabili sia per quanto riguarda i di tempi di realizzazione sia in termini di memoria necessaria ad archiviare tutti i dati. Quello che solitamente si fa è pertanto prendere un piccolo numero di template umani maschili e femminili e poi fare in modo che il calcolatore ne crei in automatico altre varianti. Tale risultato può essere conseguito o deformando alcune parti del corpo secondo coefficienti ricavati a partire da dati statistici sul tipo di popolazione in esame, oppure interpolando i template a disposizione. Quest'ultima tecnica è ad esempio stata adottata dalla Dreamworks per realizzare personaggi secondari nelle scene con folle di *Shrek*: sono stati creati alcuni corpi generici, sono stati interpolati, e al termine di questa operazione i designer hanno selezionato i modelli accettabili scartando quelli meno piacevoli esteticamente.

Una volta che si hanno a disposizione alcuni modelli, questi possono poi ulteriormente essere differenziati applicando ad essi texture e colori diversi. Basta osservare la figura 2.4 per rendersi conto che già questo semplice accorgimento migliora sensibilmente la resa visiva complessiva.

Infine, si può decidere di particolareggiare ulteriormente le persone introducendo qua e là anche qualche accessorio come occhiali, orologi, cappelli, ecc.

**Figura 2.4:** Diverse texture in vari colori applicate ad un unico template.

## 2.4 Animazione dei modelli

Altro passo importante consiste nell'animare i modelli. A questo proposito, devono essere tenuti in considerazione alcuni criteri: l'animazione deve essere efficiente da un punto di vista computazionale, deve essere compatibile con la tecnica LOD, deve consentire una certa variabilità tra un modello e un altro per creare un effetto di maggiore realismo. Per capire quali sono i problemi da affrontare, si possono considerare i due tipi di animazione più comuni nell'ambito della simulazione di folle: la camminata e la corsa.

Una prima idea che può venire in mente è quella di creare diversi cicli di locomozione da calcolare in real time per ciascun individuo. Sfortunatamente, anche un calcolatore estremamente veloce non sarebbe in grado si sostenere un simile carico di lavoro.

Un secondo approccio consiste nell'utilizzare un database di animazioni. La prima cosa da fare in questo caso è costruire un database contenente cicli di camminata che verranno poi richiamati e sfruttati dalla CPU a tempo di esecuzione. Per ottenere un buon risultato finale tale database dovrà essere creato tenendo in considerazione diverse velocità e stili di camminata. La difficoltà maggiore sta però nel reperire

i dati necessari. A questo scopo si può procedere principalmente in due modi: il primo consiste nell'ottenere i dati necessari ricavandoli in modo teorico attraverso studi di biomeccanica, cinematica diretta e cinematica inversa; il secondo si basa su tecniche di motion capture. Anche in questo caso è poi auspicabile, presi un certo numero di cicli di camminata, ottenerne di nuovi per interpolazione.

Per un effetto più realistico, è inoltre consigliabile prevedere la possibilità, per ciascuna persona, di passare da un tipo di animazione ad un altro. Naturalmente, si dovranno impiegare opportuni accorgimenti affinchè questa transizione risulti la più naturale possibile.

Nel caso sia contemplato l'utilizzo di accessori, bisognerà infine tenere conto di come la presenza di questi ultimi può influenzare le movenze della persona che li indossa (figura 2.5). Se ad esempio pensiamo che una persona cammini solitamente facendo oscillare le braccia lungo i fianchi, è ovvio che nel caso in cui questa sia impegnata in una chiamata al cellulare tenderà invece a procedere con un braccio piegato in modo da avvicinare il telefonino all'orecchio.

**Figura 2.5:** Esempi di movimenti modificati dalla presenza di accessori: mani in tasca, mani sui fianchi, chiamate telefoniche.

## 2.5 Animazione comportamentale della folla

Per alleggerire il lavoro degli animatori è importante che i componenti della folla abbiano un certo grado di autonomia nei comportamenti. Se si riescono a creare degli algoritmi in grado di decidere in modo automatico il moto della folla è ovvio che il compito dell'animatore risulterà notevolmente facilitato. Per esempio animare manualmente dieci persone che camminano in una stanza richiede dieci volte il tempo necessario ad animarne una sola, perché l'animatore si troverà a dover fronteggiare nuovi problemi quali la possibilità di collisione tra i personaggi.

In sintesi,questo paragrafo si propone di illustrare alcune delle possibile tecniche per la generazione automatica del moto e del comportamento della folla, sulla base delle caratteristiche dell'ambiente circostante e dell'eventuale presenza di altre persone. I modelli proposti per risolvere questo tipo di problema sono innumerevoli. Di seguito ne verranno citati solo alcuni per dare un'idea delle possibili strade che si possono percorrere

Il lavoro presentato da Reynolds [6] nel 1987 è considerato da molti il primo nel campo dell'animazione comportamentale. Esso descrive un metodo per animare grandi gruppi di entità chiamate **boids** (figura 2.6), che presentano un comportamento simile a quello di stormi di uccelli o branchi di pesci. Reynolds partì dal presupposto che il comportamento di gruppo è solamente il risultato dell'interazione individuale tra i componenti del gruppo stesso. Per questo motivo, sarebbe sufficiente simulare in modo ragionevolmente semplice i boidi individualmente per veder emergere un più complesso comportamento di gruppo dall'interazione tra essi.

Simulazioni di folle umane in real time furono fatte da Musse e Thalmann[7] nell'ambito di uno studio che teneva conto sia della struttura sia del comportamento della folla. Secondo gli autori la folla è strutturata come una gerarchia a tre livelli. Alcuni parametri possono essere definiti ad alto livello (per poi essere ereditati dai livelli più bassi), mentre altri possono essere decisi per una specifica struttura a livello più basso. Questo consente ad esempio di far assumere alla folla nel suo complesso un atteggiamento "felice", per poi imporre che solo uno specifico gruppo abbia un comportamento "triste". Il gruppo è la struttura più complessa fra quelle che compongono il modello: i membri del gruppo condividono il processo deci-

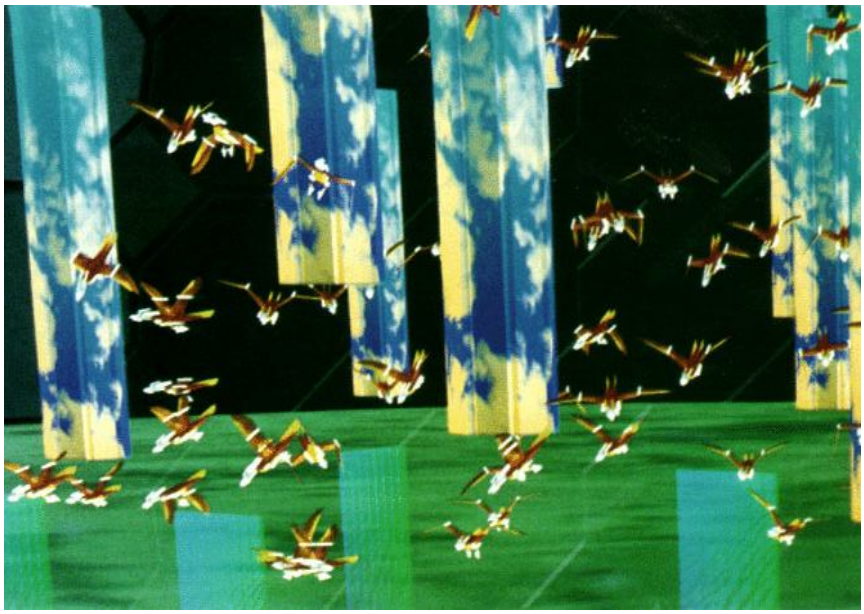

**Figura 2.6:** I "boids" di Reynolds.

sionale, ossia mettono in comune le informazioni a loro disposizione per scegliere quali azioni eseguire nel corso della simulazione; inoltre il gruppo può prendere decisioni come quella di sciogliersi o di cambiare leader. Per quanto riguarda il comportamento, i gruppi possono avere tre diversi livelli di autonomia:

- i gruppi **autonomi** si comportano in un certo modo in risposta agli eventi, in accordo con le regole comportamentali definite dall'animatore;
- i gruppi **programmati** si comportano seguendo uno script che definisce le azioni che dovranno essere compiute durante la simulazione;
- i gruppi **guidati** sono interattivamente comandati dall'utente durante l'esecuzione della simulazione.

Un altro modello per simulare folle di umani virtuali fu presentato da Ulicny e Thalmann[8] nel 2002. Questo modello si concentra sul comportamento individuale degli agenti anziché su quello del gruppo. Il modello comportamentale si compone di tre livelli:

- Il livello più alto è composto da una serie di regole. Ogni regola ha una precondizione (che specifica a quali agenti è consentito eseguire tale regola e

sotto quali circostanze) e una post-condizione (che descrive gli effetti dell'esecuzione della regola stessa). L'esecuzione delle regole può modificare il comportamento corrente dell'agente, cambiarne gli attiributi o scatenare eventi.

- Il comportamento corrente dell'agente rappresenta il livello intermedio, in cui ogni possibile comportamento è visto come parte di una macchina a stati finiti, in base alla quale vengono determinati i comportamenti di basso livello.
- Le azioni vere e proprie eseguite istante per istante dall'agente rappresentano il livello più basso. Tali azioni, oltre che essere influenzate dal livello superiore, sono definite anche andando ad eseguire algoritmi di path-planning e collision avoidance.

## 2.6 Pianificazione delle traiettorie

Data la posizione iniziale di un pedone, la destinazione finale desiderata e la descrizione geometrica dell'ambiente, come calcolare un percorso che congiunga questi due punti evitando collisioni con gli ostacoli? Il problema che prendiamo in esame parlando di pianificazione di traiettorie è esattamente questo. Vediamo di seguito alcune delle principali soluzioni che si possono trovare in letteratura.

### 2.6.1 Soluzioni proposte nel mondo della robotica

Il problema della pianificazione di traiettorie è da tempo diventato oggetto di studio da parte della robotica, al fine di garantire ai robot autonomia di movimento anche in ambienti contenenti ostacoli. Gli approcci sviluppati dalla robotica non sono direttamente applicabili al campo della simulazione di folle, tuttavia è possibile pensare a metodi simili basati sugli stessi principi. Di seguito ne vengono riportati alcuni fra i tanti che sono stati proposti.

I metodi discreti sono probabilmente i più semplici da mettere in pratica. L'idea è di usare una rappresentazione discreta dell'ambiente, immaginando che sul pavimento sia distesa una griglia. A questo punto, ad ogni cella della griglia è associata una variabile di stato che indica se la cella è occupata da un ostacolo oppure no. Il

movimento è consentito solo tra celle libere adiacenti. A questo punto il problema si riduce a selezionare la cella libera adiacente verso cui dirigersi ricorrendo ad un algoritmo *ad hoc*.

Un altro approccio ha come idea chiave quella di rimanere il più possibile lontano dagli ostacoli in modo da poter navigare in modo sicuro. Un'apposita mappa segnalerà dunque la distanza rispetto all'ostacolo più vicino per ogni punto dello spazio. L'unione dei punti di massimo di tale mappa rappresenterà l'insieme dei percorsi più sicuri per evitare collisioni.

A differenza dei metodi visti, i metodi reattivi considerano il problema solo localmente. A partire da un'osservazione locale dell'ambiente, ad ogni passo viene computata una nuova azione. I metodi reattivi sono generalmente semplici da implementare, ma poichè l'ambiente è preso in considerazione solo localmente possono portare a dei punti morti. Per questo motivo essi possono anche essere utilizzati solo come una legge di controllo di feedback: un percorso completo viene calcolato attraverso un metodo di tipo globale, mentre il metodo reattivo viene impiegato solo per calcolare il tragitto da compiere localmente.

### 2.6.2 Pianificazione di traiettorie per folle

Il problema della pianificazione di traiettorie per folle è stato risolto in molti modi diversi. Le strategie proposte dalla robotica sono state prese come punto di partenza per le soluzioni trovate, con adattamenti *ad hoc* per i problemi specifici delle folle.

Nell'ambito delle applicazioni per gli studi di sicurezza trovano solitamente impiego tre principali classi di approcci:

- i modelli **flow-based** affrontano il problema di simulare la folla da un punto di vista globale: i pedoni non sono modellati come entità individuali, ma come facenti parte di un flusso. Richiamandosi dunque anche a principi di fisica (con opportuni adattamenti) si definiscono quindi le proprietà fisiche del flusso e se ne studia il comportamento a partire da queste.

- negli approcci ad **automi cellulari** l'ambiente è modellato come una griglia di celle. Le celle sono occupate da pedoni e la densità locale, il flusso e la

velocità sono tenuti in considerazione per calcolare il movimento dei pedoni di cella in cella.

- nei modelli **ad agenti** una mappa delle distanze simile a quella utilizzata nell'ambito robotico viene vista come un campo vettoriale, e i pedoni seguono il gradiente per raggiungere la loro destinazione[2].

Nel campo delle applicazioni destinate all'intrattenimento la necessità primaria è invece l'interattività, dove il termine "interattività" assume un significato leggermente diverso a seconda dello specifico impiego. Ad esempio, l'industria cinematografica si serve di simulazioni di folle per ottenere suggestive immagini con migliaia di attori virtuali: la simulazione viene solitamente precalcolata, tuttavia deve essere disponibile in tempi ragionevoli per consentire di fare diverse prove. Per raggiungere questo risultato Massive (il software più utilizzato in questo ambito, di cui si è già parlato precedentemente), fornisce ad ogni agente una propria intelligenza artificiale, una sorta di cervello indipendente che gli consente di eseguire la propria navigazione in modo autonomo.


**Figura 2.7:** Age of Empires III, uno dei più famosi giochi di strategia per PC.

[2]Legion, di cui si è parlato nel paragrafo 2.2 è ad esempio un simulatore che si serve di un modello basato su agenti.

Anche per i video game, sopratutto per quelli di strategia real-time che prevedono un elevato numero di entità (figura 2.7), l'interattività è cruciale. Tuttavia in questo caso il tempo computazionale deve essere necessariamente piccolo se si vogliono ottenere prestazioni real-time: il giocatore deve poter vedere immediatamente le conseguenze dei suoi comandi senza discontinuità nel gioco. Tra i vari approcci possibili possiamo citare quello di Kamphuis e Overmars[9], che hanno specializzato il loro pianificatore di traiettorie in modo da considerare gruppi di pedoni. L'idea è di pianificare il movimento per una superficie deformabile contenente tutti i membri dello stesso gruppo. L'area della superficie rimane sempre uguale in modo che le persone abbiano sempre abbastanza spazio per potersi muovere pur rimanendo raggruppate. La superficie si può deformare in modo che anche i passaggi più stretti possano essere attraversati.

Se si ha a che fare con applicazioni di realtà virtuale diventa invece importante coniugare realismo e performance. Anche in questo caso, le soluzioni proposte sono innumerevoli. Tra esse si può ricordare quella di Tecchia e al.[10], che sfrutta una successione di layer (mappe) per controllare il comportamento dei pedoni. Data l'attuale posizione del pedone, il metodo analizza una dopo l'altra le informazioni di ciascun layer al fine di decidere la nuova posizione. Uno strato è dedicato alla collision detection (rispetto a ostacoli fissi ed altri pedoni), uno definisci i comportamenti da tenere o gli obbiettivi da raggiungere in quella particolare zona, e l'ultimo determina le aree di attrazione (goal). Secondo gli autori, la combinazione di questi tre layer è sufficiente per produrre un movimento di folla realistico.

Infine, un'ultima tecnica molto nota e utilizzata è quella dei **grafi di navigazione** [11]. Dato l'ambiente che si vuole popolare, la prima cosa che si va a fare è identificare le aree navigabili, ossia quelle prive di ostacoli e in cui la pendenza sia sufficientemente piccola. A questo punto, nelle aree praticabili, vengono tracciati dei cerchi in modo che il loro diametro sia massimo. È consentito passare da un cerchio a un altro solo se i due si intersecano. I segmenti che congiungono i punti in cui due cerchi adiacenti si intersecano prendono il nome di **gate**. Congiungendo tra di loro gli estremi dei vari gate si ottengono dei **corridoi** attraverso cui è permesso il passaggio (v. figura 2.8). Da notare che i grafi di navigazione sono calcolati solo una volta per ogni ambiente: una volta che questo è stato fatto, a seconda del punto di partenza del pedone e della posizione del suo obbiettivo, si utilizza un algorit-

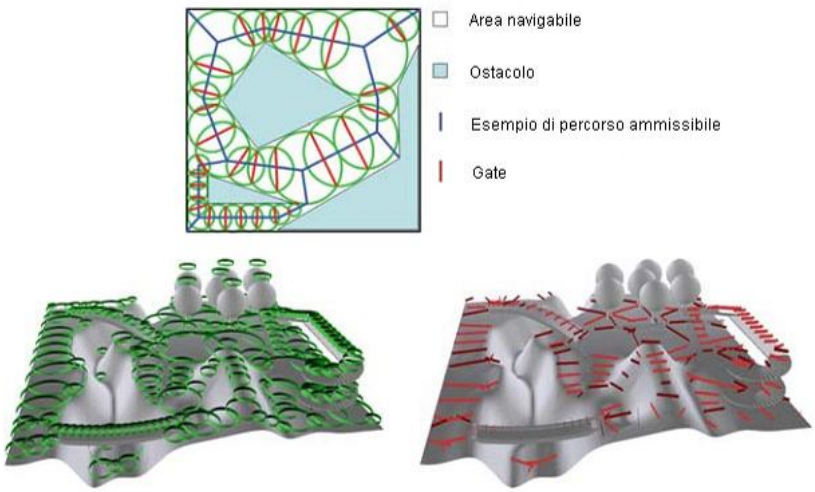


**Figura 2.8:** Costruzione dei grafi di navigazione.

mo di ricerca su grafo (ad esempio quello di Dijkstra) per decidere quali corridoi il pedone dovrà percorrere.

## 2.7 Correlazione tra folle virtuali e folle reali

Se si vuole ottenere una folla il più possibile realistica la soluzione migliore è simularla partendo da dati ricavati a partire dall'osservazione di folle reali. L'osservazione di ambienti reali popolati da folle può consentire di individuare diversi aspetti di interessa, tra cui l'utilizzo dello spazio (le aree che vengono maggiormente occupate, le traiettorie principalmente seguite, le zone nelle quali viene eseguita una certa azione), la densità della folla (la distanza tra gli individui e la dimensione della folla stessa), la struttura della folla (considerando la presenza di gruppi[3]) e i tipi di comportamenti tenuti dalle persone (camminata, corsa, assunzione di determinate posture, ecc.). Altro aspetto interessante è cercare di capire cosa accade al verificarsi di particolari eventi, come l'aprirsi delle porte di un treno o lo scoppio di un incendio. Si può infine tenere conto di alcuni aspetti di carattere più sociologico

[3]Per un'interessante analisi delle dimensioni e della struttura tipica dei gruppi di persone si rimanda alla voce [12] della bibliografia.

come la presenza di gerarchie tra le persone, di forze di attrazione o di repulsione tra certi individui, della tendenza dei gruppi a disgregarsi/riaggregarsi e a seguire un leader.

Per ricavare tutti questi dati ci si può servire delle registrazioni di una o più telecamere. Tuttavia, è chiaro che estrapolare i dati manualmente risulterebbe un lavoro estremamente lungo e complesso. Fortunatamente, ci si può avvalere di strumenti automatici di identificazione di pedoni e folle basati su tecniche di visione artificiale. Esistono molteplici algoritmi finalizzati al *people tracking* (figur 2.9). Fra questi i più noti sono probabilmente quelli di *background removal*. In poche parole, si procede ad estrapolare un modello matematico dello sfondo, che viene poi comparato con ciascun frame della videosequenza. A questo punto, i pixel con una sufficiente discrepanza rispetto a quelli dello sfondo vengono considerati come in primo piano: un insieme di pixel connessi prende solitamente il nome di **blob**. Dall'analisi dei blob e dei loro spostamenti è possibile ricavare informazioni sulla presenza di persone nella scena, sulle traiettorie da esse seguite, sulle dimensioni dei gruppi che si creano, ecc.

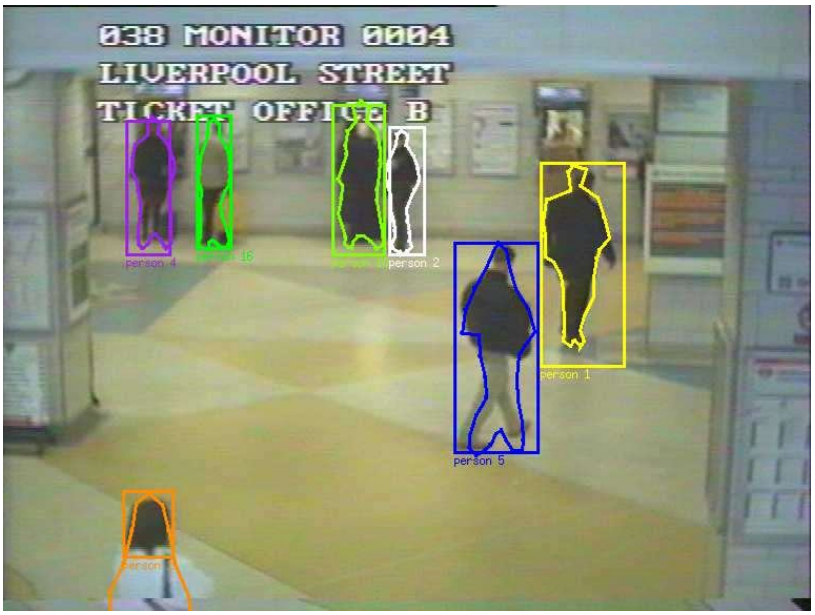


**Figura 2.9:** Esempio di people tracking per un sistema di videosorveglianza.

## 2.8 Level Of Detail

La tecnica LOD (Level Of Detail) è molto usata nell'ambito della simulazione di folle. Avendo a che fare con migliaia di persone e dovendo processare una grande quantità di informazioni per ciascuna di esse, ne consegue che il carico complessivo di lavoro risultante è molto pesante anche per i moderni processori. Per alleggerire il carico computazionale richiesto in simulazioni particolarmente complesse, si può ricorrere alla tecnica LOD[4]. L'idea di fondo è quella di rappresentare gli oggetti più lontani con un livello di dettaglio (e quindi con un numero di poligoni) minore, in modo da ridurre i tempi di calcolo, giocando sul fatto che gli oggetti più piccoli e lontani non possono comunque essere distinti chiaramente dall'occhio umano, per cui la differenza rispetto ai modelli più dettagliati risulta visivamente trascurabile.

Applicando questo principio alla simulazione di folle, si può pensare di raffigurare ogni persona con una rappresentazione specifica a seconda della sua posizione rispetto alla telecamera (vedi figura 2.10), ottenendo un compromesso tra costo di rendering e qualità. A questo scopo, è necessario introdurre brevemente il concetto di **template** umano e le sue possibili rappresentazioni, che entreranno in gioco nella tecnica LOD.

Una qualsiasi categoria di persona (uomo, donna o bambino) può essere descritta da un template umano, che consiste di:

- Uno scheletro, composto da giunti che rappresentano le articolazioni;
- Un insieme di mesh, ciascuna rappresentante lo stesso umano virtuale ma con un numero di triangoli decrescente;
- Un set di sequenze di animazione che il template può eseguire.

A partire dal template appena descritto è possibile introdurre il concetto di **mesh deformabile**. Si tratta di una superficie formata da un insieme di triangoli (o più in generale poligoni) che racchiude uno scheletro: quando lo scheletro si muove, i vertici della mesh seguono il suo movimento, similmente a quello che fa la pelle (si parla di *animazione scheletale*). Ogni vertice della mesh è influenzato in modo opportuno dai giunti più o meno vicini. Sfortunatamente, utilizzare solo mesh

[4]Per approfondire l'argomento si veda [13].


**Figura 2.10:** Diverse rappresentazioni di un template umano in tecnica LOD.

deformabili per una folla molto numerosa sarebbe proibitivo, per cui nella tecnica LOD esse vengono impiegate solo per i soggetti più vicini alla telecamera. Va osservato che per creare una mesh sono necessari dei designer piuttosto abili, ma una volta finita questa può essere facilmente usata per derivarne le successive rappresentazioni del template, ossia le mesh rigide e gli impostori.

Una **mesh rigida** (figura 2.11) è una postura geometrica precalcolata di una mesh deformabile (l'aspetto esteriore è dunque il medesimo). Per realizzare le animazioni, si va a calcolare la corrispondente mesh rigida per ogni fotogramma chiave dell'animazione scheletale. In altre parole, ogni vertice viene deformato dalla CPU a seconda dei movimenti dello scheletro, dopodichè il risultato viene memorizzato in una tabella di vertici, normali (punti 3D) e coordinate di punti per le texture (punti 2D). A tempo di esecuzione poi, l'animazione viene eseguita semplicemente come successione di posture precalcolate. I vantaggi sono la maggior velocità di esecuzione dell'animazione (poichè la deformazione dello scheletro e dei vertici della mesh è già stata calcolata) e la ridotta necessità di comunicazione tra CPU e GPU (non devono essere inviate trasformazioni di giunto); inoltre l'aspetto è graficamente identico a quello di un'animazione scheletale. Tuttavia, le mesh rigide devono essere mostrate lontano dalla telecamera, perché con questa tecnica alcuni tipi

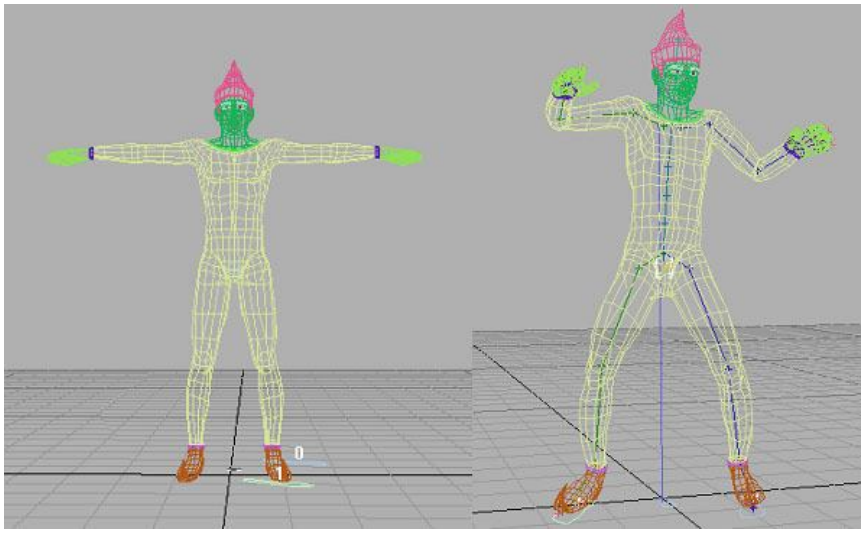

**Figura 2.11:** Esempio di mesh rigida resa deformabile con l'aggiunta di uno scheletro interno.

di animazione, come quelle delle mani o della faccia non sono possibili.

Gli **impostori** sono la rappresentazione meno dettagliata ma più intensamente sfruttata nell'ambito del rendering di folle. Un impostore rappresenta un umano virtuale servendosi solo di un poligono. Ciò è sufficiente a dare l'illusione della presenza di una persona a grande distanza dalla telecamera. Similmente a quanto detto per le mesh rigide, un'animazione basata su impostori è una successione di keyframe 2D estratti dall'animazione scheletale originaria. La differenza rispetto alle mesh rigide è che in questo caso abbiamo solo una rappresentazione 2D di ogni postura, non l'intera geometria. Siccome però gli impostori sono 2D, è necessario memorizzare cordinate delle normali e delle texture da molti punti di vista, in modo che a a tempo di esecuzione, quando la telecamera ruota, all'impostore possano essere applicate le giuste texture a seconda dei punti di vista. Il principale vantaggio degli impostori è che sono molto efficienti, in quanto è necessario un solo poligono per ciascun umano raffigurato. Per questo essi costituiscono spesso la maggior parte della folla. Tuttavia, la qualità di rendering è scarsa, e pertanto non possono essere impiegati vicino alla telecamera. Inoltre, lo spazio in memoria necessario per immagazzinare un impostore è molto elevato (figura 2.12), a causa del grande numero

di texture che devono essere salvate.

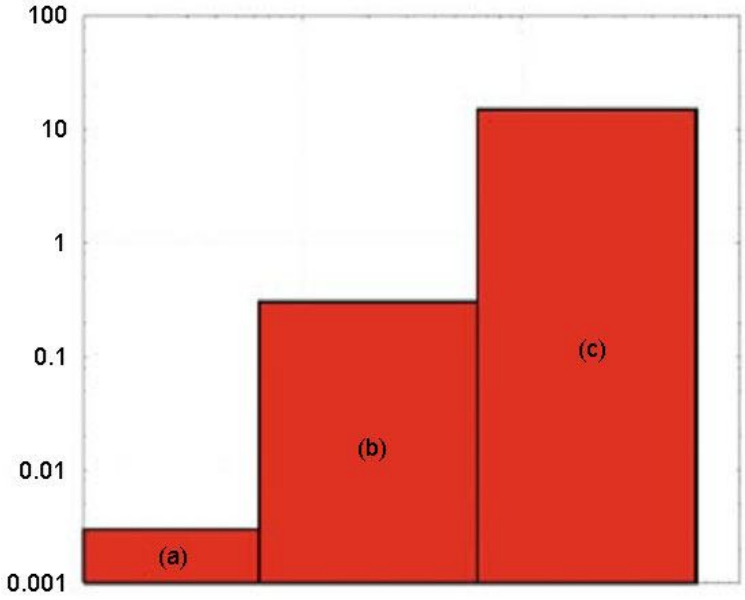


**Figura 2.12:** Spazio di archiviazione (in Mb) per un secondo di una clip di animazione per (a) una mesh deformabile, (b) una mesh rigida, (c) un impostore.

# Capitolo 3

# Strumenti software utilizzati

## 3.1 Horde3D

Horde3D [14] è un software di simulazione grafica open source per l'animazione e il rendering 3D basato su OpenGL. Si tratta di un progetto supportato dall'università di Asburgo e distribuito sotto licenza LGPL. E' concepito per poter competere in qualità e performance con gli engine commerciali di nuova generazione, pur rimanendo leggero e concettualmente chiaro. Horde3D (figura reflogoHorde) segue una strategia di integrazione diversa da quella di altri noti engine grafici quali OGRE 3D [15] o Irrlicht [16]. Questi forniscono una libreria di classi object-oriented a partire dalle quali l'utente può derivare le proprie implementazioni, mentre le funzionalità di Horde3D sono accessibili tramite un'interfaccia procedurale simile a quella delle API Microsoft Win32. Questa semplice interfaccia stile C ha il vantaggio di essere più facile da leggere e da imparare, perché mette a disposizione solo ciò che è realmente necessario all'utente. Un altro beneficio è la maggiore portabilità: infatti molteplici wrapper sono già stati scritti per consentire l'utilizzo di Horde3D con C#, Java, LUA e altri linguaggi. In particolare, per la realizzazione di questo progetto è stato utilizzato il linguaggio C++.

L'interfaccia procedurale non impedisce di utilizzare il motore con linguaggi orientati agli oggetti. Internamente Horde3D stesso è strettamente object-oriented. Gli oggetti trattati dal motore, quali modelli, luci, materiali, shader e così via sono messi a disposizione dell'applicazione attraverso dei gestori chiamati **handles**. Un


**Figura 3.1:** Logo di Horde3D.

handle è concettualmente simile a un puntatore. Horde3D ha specifiche funzioni per creare oggetti che restituiscono un handle all'oggetto costruito o un "NULL-handle" (fondamentalmente l'equivalente di un NULL-pointer) in caso di insuccesso. L'handle deve essere memorizzato dall'applicazione e può poi essere utilizzato per accedere all'oggetto al fine di modificarne le proprietà o di rilasciare la relativa memoria.

Un particolare punto di interesse di questo motore grafico è che è stato progettato per consentire il rendering di folle di grandi dimensioni con una buona qualità. Le possibilità di utilizzo di Horde3D non sono limitate a questo tipo di applicazione, tuttavia ci sono alcune caratteristiche che lo rendono particolarmente adatto ad esso. Ad esempio, i dati relativi alle risorse grafiche sono internamente memorizzati in modo da rendere più veloci le animazioni, la geometria è ottimizzata per un uso efficiente della cache ed è possibile utilizzare diversi livelli di dettaglio per un modello. Inoltre, si può pensare di usare tecniche di vertex skinning [1] e di deferred shading [2] al fine di velocizzare il rendering di scene con molte luci.

Horde3D può essere utilizzato in ambiente Windows, Linux e Mac OS X. Se si sceglie di operare sotto Windows, come è stato fatto per questa tesi, è necessario lavorare in ambiente Visual Studio. L'attuale package per Horde3D SDK contiene al suo interno un solution file per Microsoft Visual Studio 2005, tuttavia l'SDK può

[1] Con il termine *vertex skinning* intendiamo il processo di creazione di un legame tra i vertici di una mesh poligonale e le ossa di uno scheletro. Quando le ossa subiscono una trasformazione, questa viene applicata anche ai vertici della mesh, proporzionalmente al peso che un determinato osso esercita su ciascun vertice.

[2] Con il termine *deferred shading* si indica una tecnica di shading tridimensionale nella quale il risultato dell'algoritmo di shading è calcolato dividendo quest'ultimo in parti più piccole il cui risultato viene memorizzato in buffer intermedi e solo successivamente combinato insieme.

essere usato anche con Visual C++ Express Edition e Visual Studio 2008 (convertendo la soluzione in modo appropriato con il conversion wizard del programma). Tutto ciò che bisogna fare per iniziare a lavorare è aggiungere le librerie in figura 3.2.

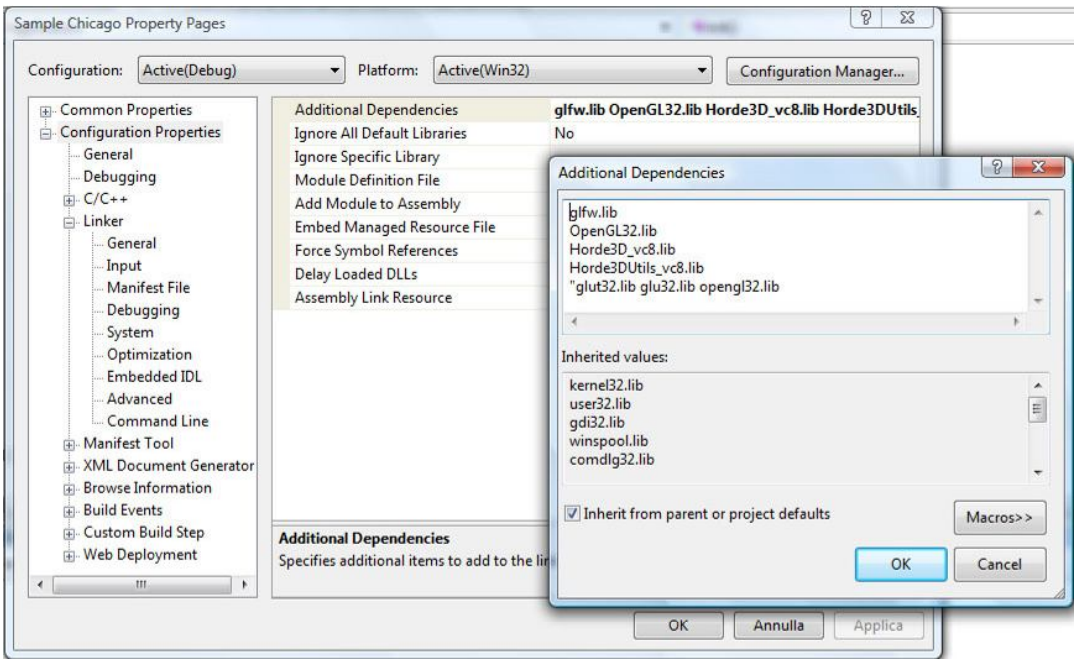


**Figura 3.2:** Librerie da includere per utilizzare Horde3D SDK in Visual Studio. Per aggiungerle, selezionare *Project->Project Properties->Linker->Input->Addictional Dependencies*.

## 3.2 ColladaConverter

Ogni engine grafico avanzato richiede una metodologia per importare risorse da software per la modellazione 3D all'interno dell'engine stesso e delle sue applicazioni. Tradizionalmente è necessario utilizzare strumenti di conversione ed esportazione differenti per ciascuno strumento di modellazione. Per superare questo problema, si è pensato di introdurre uno standard per lo scambio di risorse. Horde3D utilizza COLLADA (*COLLAborative Design Activity*), un formato 3D standard su

file XML. Creato da Sony Entertainment, è presto divenuto uno standard oggi mantenuto dalla società Kronos Group, che ne distribuisce gratuitamente il codice sorgente. Si tratta di un formato ormai previsto in tutti gli engine 3d, software di modellazione/cad/rendering e convertitori su ogni piattaforma e sistema operativo (console, quali ad esempio PlayStation 3, comprese).Per maggiori dettagli, si veda la sottosezione 3.2.1.

E' basilare riconoscere COLLADA come un mero formato di interscambio: non è possibile utilizzarlo per modelli finali per videogame o software 3D. Affinchè il modello possa essere inserito nel motore o altro, esso dovrà essere convertito nel suo formato definitivo. Horde3D SDK contiene uno strumento di conversione chiamato Collada Converter, che consente di convertire risorse dal formato Collada 1.4 in un formato specifico per Horde3D. In questo modo non solo è possibile esportare mesh poligonali, ma è possibile definire un'intera scena, integrando tutte le caratteristiche per l'animazione, shader, fisica.

Per importare un modello in Horde3D la prima cosa da fare è esportarlo dal proprio software di modellazione in formato Collada. Per questa tesi in particolare sono stati utilizzati Google Sketchup Pro e 3ds Max 9: Google Skatchup Pro presenta già un apposito tool di conversione al suo interno, mentre per quanto riguarda 3ds Max 9 è necessario installare il plugin ColladaMax [17]. In questo modo si ottiene un file con estensione .DAE, che dovrà successivamente essere trascinato sul file *Horde3D\Binaries\Win32\ColladaConv.exe*: questa operazione genererà due cartelle *animations* e *models*. Ora è sufficiente copiare il contenuto di tali directory rispettivamente nelle cartelle *Horde3D\Binaries\Content\Animations* e *Horde3D\Binaries\Content\Models*.

Per il momento esistono alcune restrizioni riguardo ai file Collada compatibili con il convertitore per Horde3D: tutta la geometria deve essere memorizzata attraverso triangoli e le animazioni devono essere esportare sotto forma di sampled keyframe data.

### 3.2.1 Il formato COLLADA

Come si è detto, COLLADA è un formato standard per lo scambio di risorse per il rendering e la modellazione 3D tra piattaforme e applicazioni di natura diversa.

Le software house tendono solitamente ad utilizzare formati proprietari, tenendo nascosta la struttura con cui i dati utilizzati dalle loro applicazioni vengono memorizzati. In questo modo, l'utente necessita di un qualche tipo di interfaccia per accedere ai dati stessi ed è quindi vincolato a continuare ad usare gli strumenti che il produttore mette a disposizione. Le risorse create con un particolare software non possono essere lette da programmi di altri produttori, per cui il costo di cambiare strumento diventa proporzionale alla quantità di contenuti che lo sviluppatore ha generato in quel formato. Le software house creano in questo modo nei clienti una sorta di "dipendenza" dai loro prodotti. COLLADA nasce dall'esigenza di superare questo tipo di problematiche.

COLLADA è una tecnologia che codifica sia dei contenuti, sia l'informazione su come questi contenuti dovranno essere utilizzati (semantica). Presenta una struttura ben definita, in modo da organizzare contenuti complessi in unità più semplici facilmente gestibili. Inoltre, COLLADA si serve di una codifica che faccia uso solo di caratteri internazionali, al fine di essere leggibile ed utilizzabile in ogni parte del mondo.

Il linguaggio XML (*eXtensible Markup Language*) è stato scelto per COLLADA perché presenta tutte le caratteristiche richieste. XML è ben definito dal World Wide Web Consortium (W3C [18]) ed è supportato da un gran numero di strumenti e applicazioni. Rappresenta un linguaggio standard per descrivere la struttura, la semantica e il contenuto di un documento.

Il formato COLLADA è definito da:

- una specifica formale, scritta in linguaggio XML, di quali dovranno essere la struttura e i contenuti di ogni documento che aderisca a questo formato (**COLLADA schema**);
- una documentazione leggibile dall'utente che illustra, partendo dallo schema, ciascun elemento di quest'ultimo e fornisce linee guida aggiuntive.

Per i file COLLADA è raccomandata l'estensione ".dae" (Digital Assets Exchange), ma sarebbe anche corretta (sebbene sconsigliata) l'estensione ".xml". In realtà però, un documento COLLADA non deve necessariamente essere archiviato in un file: potrebbe anche essere memorizzato all'interno di un database o essere richiamato dall'applicazione attraverso una richiesta HTTP. Ciò garantisce che

questo standard sia fruibile da applicazioni di qualsiasi tipo, senza limitazioni sulla modalità con cui essere accedono ai propri dati.

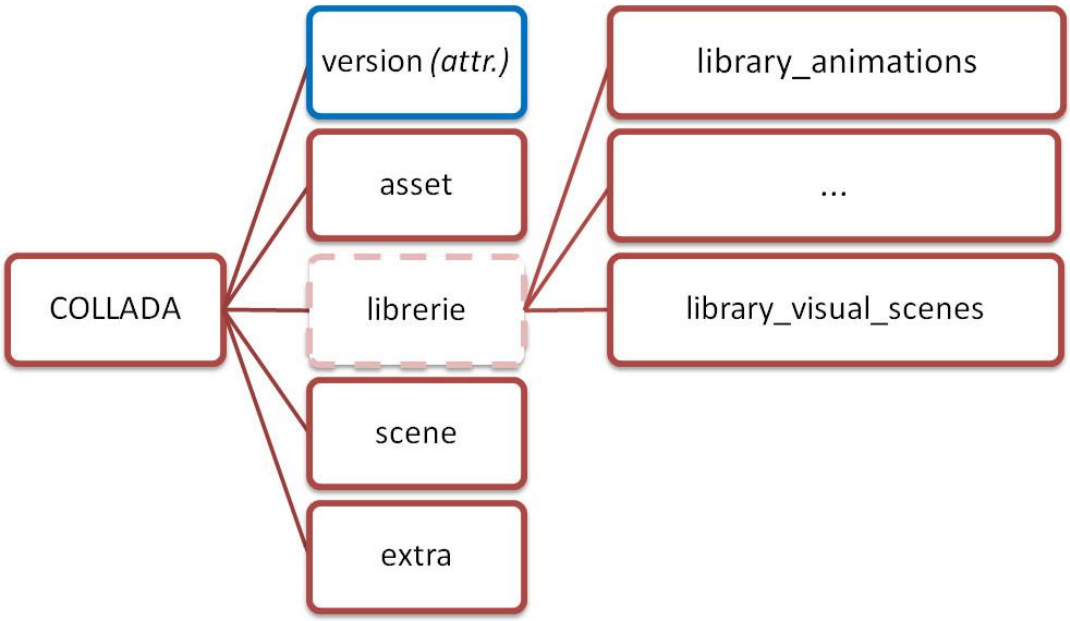


**Figura 3.3:** Struttura di un documento COLLADA.

Un documento XML ha tipicamente una struttura gerarchica, in cui una serie di elementi diversi sono annidati l'uno dentro all'altro. L'elemento `<COLLADA>` è la radice di ogni documento COLLADA. Un elemento `<COLLADA>` deve contenere un elemento `<asset>`. Gli altri elementi figli sono opzionali. Gli elementi figli possibili sono elencati di seguito:

- deve sempre obbligatoriamente essere presente un elemento `<asset>`, che deve essere unico per ciascun documento;
- ci possono essere zero o più elementi libreria, indicati come `<library_...>`;
- può opzionalmente essere presente un elemento `<scene>`, che deve essere unico e indica la scena di default da utilizzare per quel documento;
- possono opzionalmente essere presenti più elementi `<extra>`.

L'ordine con cui tali elementi compaiono deve essere obbligatoriamente questo.

Per l'elemento <COLLADA> viene inoltre solitamente definito un attributo `version`, che indica la versione dello standard adottata. La tipica struttura di un documento COLLADA è mostrata in figura 3.3.

## 3.3 Google SketchUp Pro 7

Google SketchUp [3] è un altro software di modellazione 3D, disponibile per le piattaforme Windows e Mac. SketchUp è un software versatile, potente e nel contempo semplice da imparare e da utilizzare. Permette la creazione di forme bidimensionali e tridimensionali in modo semplice, intuitivo e veloce, fornendo al disegnatore uno strumento in grado di assisterlo dal punto di vista grafico e di consentirgli un'esplorazione dinamica e creativa degli oggetti, dei materiali e della luce. Il 9 gennaio 2007, la software house che lo aveva inizialmente prodotto è stata acquisita da Google, poiché l'applicativo è particolarmente indicato per modellare gli edifici da visualizzare con Google Earth.

Sketchup è più semplice da utilizzare ma anche molto meno ricco di funzionalità rispetto a 3ds Max. Tuttavia, si è scelto di usarlo per facilitare il processo di creazione degli elementi architettonici presenti nello scenario di esempio proposto nel simulatore di folle sviluppato con questa tesi. Tramite uno speciale comando è possibile scaricare gratuitamente modelli di qualsiasi genere direttamente nella propria scena, scegliendoli all'interno di un vasto archivio online (figura 3.4). Nell'ambito di questa tesi si sono quindi cercati in rete modelli già pronti di edifici con un basso numero di poligoni; essi sono poi stati differenziati attraverso l'utilizzo di texture diverse, esportati in formato Collada (tramite l'opportuna voce di menu *File->Export*) ed infine importati, opportunamente scalati, nello scenario del simulatore.

[3] per scaricare il software collegarsi al sito http://sketchup.google.it/download.html

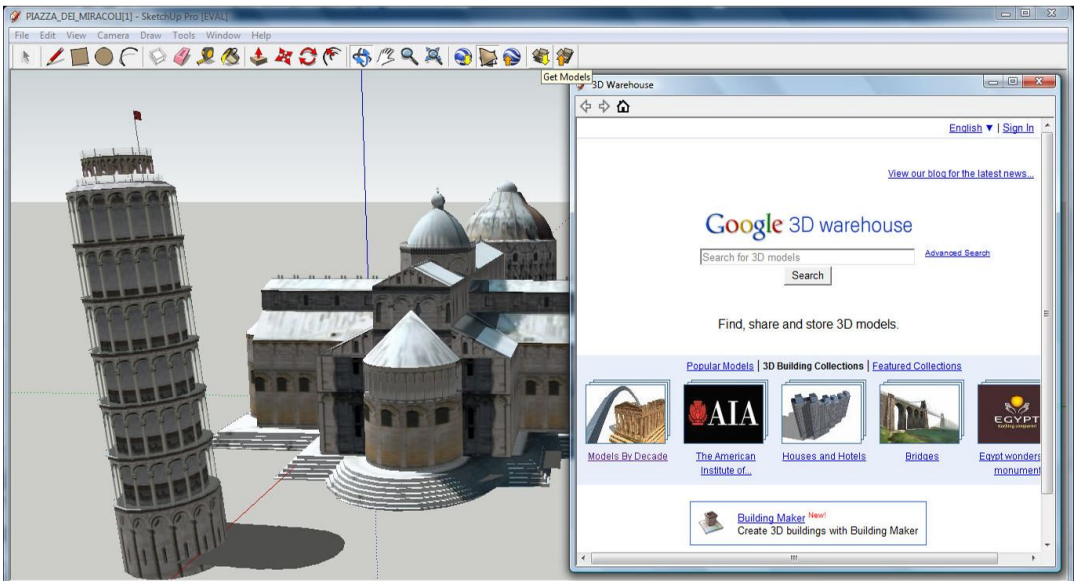


**Figura 3.4:** Google Sketchup Pro ed il suo archivio di modelli online

## 3.4 Autodesk 3ds Max 9

I personaggi animati utilizzati in questo progetto sono stati realizzati con 3ds Max 9 (figura 3.5). 3ds Max è un programma di grafica vettoriale tridimensionale e animazione, realizzato dalla divisione Media & Entertainement di Autodesk. Era stato originariamente sviluppato come successore di 3D Studio per DOS, ma su piattaforma Win32. Si tratta di uno dei più utilizzati software per la creazione di modelli 3D per numerose ragioni, tra cui le potenti capacità di editing e la sua architettura basata su plugin. Infatti anche se molti strumenti non sono parte del prodotto, esso dispone di una grande scelta di plugin realizzati da terze parti.

In particolare, nell'ambito della tesi è stato utilizzato il plugin **ColladaMax** che consente di esportare modelli realizzati con 3d Studio Max in un formato .dae adatto per Horde3D. La versione più recente di questo prodotto è la 10, tuttavia si è reso necessario ricorrere a quella precedente in quanto ColladaMax non è ancora disponibile per versione 10. In realtà 3ds Max (sia nella versione 9, sia nella 10)

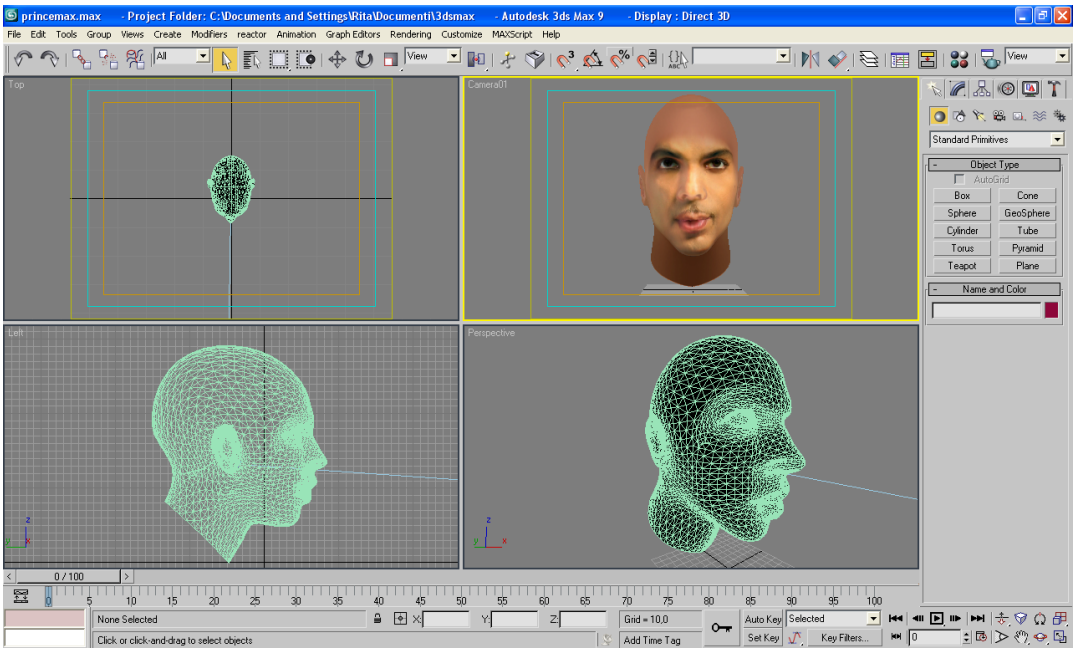


**Figura 3.5:** Esempio di schermata di 3dsMax

presenta già al suo interno uno strumento che permette di esportare file in formato Collada, però esso presenta delle imperfezioni (per esempio non vengono esportate correttamente le animazioni, che non vengono quindi caricate da Horde3D). Nel corso di questo lavoro di tesi, questo software è stato usato per dotare di uno scheletro interno e successivamente animare alcuni modelli statici tridimensionali di persone umane. Il processo seguito sarà descritto nel capitolo 4.

## 3.5 CubeMapGen

Vediamo infine come è stato possibile realizzare il panorama che circonda la scena del nostro progetto. L'ambiente in cui i personaggi si muovono è racchiuso all'interno di quello che viene chiamato uno **skybox**. Uno skybox è un metodo utilizzato nei videogame per far sembrare il paesaggio circostante più grande di quanto in realtà esso sia. La scena è racchiusa in un cubo: il cielo, le montagne in lontananza,

gli edifici distanti e altri oggetti irraggiungibili sono disegnati sulle facce del cubo, creando nell'osservatore l'illusione di essere circondati da entità tridimensionali. Questo effetto viene ottenuto mappando su ciascuna delle 6 facce del cubo differenti texture ordinate in modo opportuno. L'insieme di queste texture prende il nome di **cubemap**. Per la disposizione delle immagini, Horde3D segue la convenzione mostrata in figura 3.6.

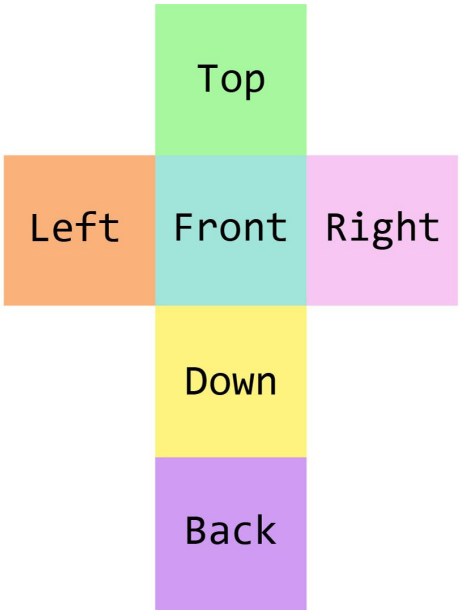


**Figura 3.6:** La convenzione per le cubemap seguita da Horde3D

In particolare, per questa tesi è stato riprodotto uno spazio aperto con delle montagne in lontananza (figura 3.7).

Horde3D accetta cubemap in formato **DirectDraw Surface** (**.dds**). Tale formato, creato da Microsoft, è appositamente pensato per essere utilizzato in applicazioni di real time rendering (come i videogiochi 3D), dove viene impiegato per memorizzare texture e cubemap entrambe con o senza compressione. Per convertire la cubemap scelta in formato .dss è stato necessario ricorrere ad un software freeware (sviluppato da *ATI 3D Application Research Group*) chiamato **CubeMapGen**. Si tratta di uno strumento pensato per importare, creare, visualizzare ed esportare cubemap. All'apertura il programma si presenta come in figura 3.8.

Sulla destra si trovano una serie di pulsanti che ci permettono di compiere varie azioni, mentre sulla sinistra sarà visualizzato il risultato delle azioni compiute. Al-

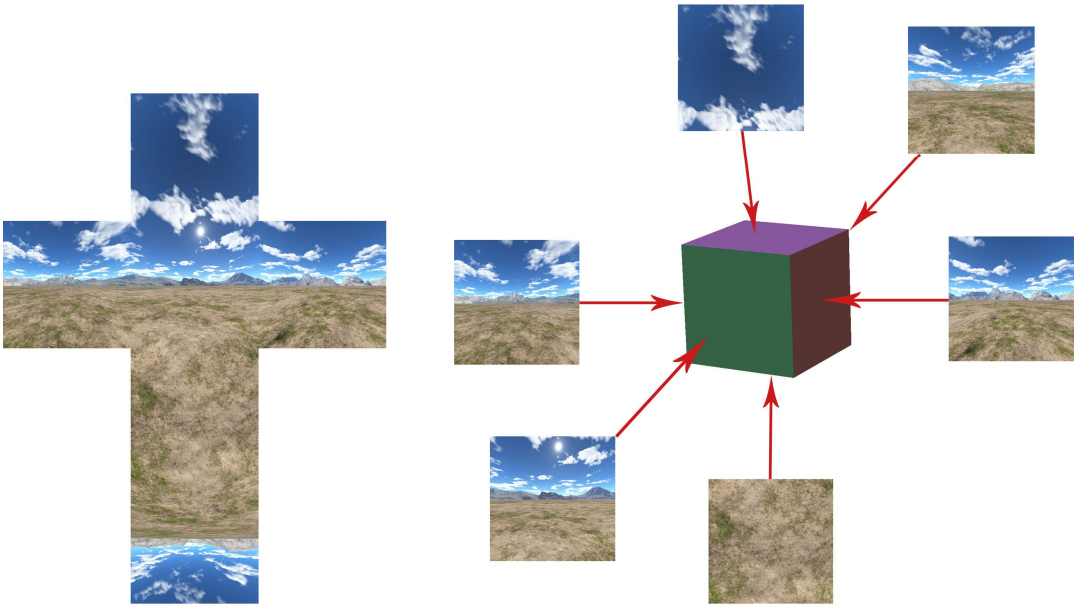

**Figura 3.7:** La cubemap utilizzata nel nostro progetto

l'inizio si vedono solo uno sfondo blu con al centro una sfera di vari colori: nel momento in cui si va a caricare le immagini che faranno parte della cubemap, ciascuna di esse viene deformata e mappata in modo da occupare una delle zone colorate presenti sulla sfera. È possibile caricare un'immagine alla volta selezionando la faccia su cui si desidera agire (*Select Cubemap Face*) e poi cliccando su *Load Cube Map Face*; oppure si può acquisire una completa croce di immagini attraverso il pulsante *Load Cube Cross*. Le texture caricate verranno immediatamente visualizzate, al posto delle faccette colorate, sulla sfera di destra (figura 3.9). Selezionando poi il checkbox *Skybox* si ottiene una sorta di anteprima di come risulterà lo skybox che verrà generato con l'introduzione della cubemap scelta. Selezionado la sfera con il mouse e muovendo poi il puntatore si può infine osservare la scena a 360 gradi. Se si è soddisfatti del risultato, è possibile esportare la cubemap in formato .dss attraverso il il pulsante *Save Cubemap .dds*.

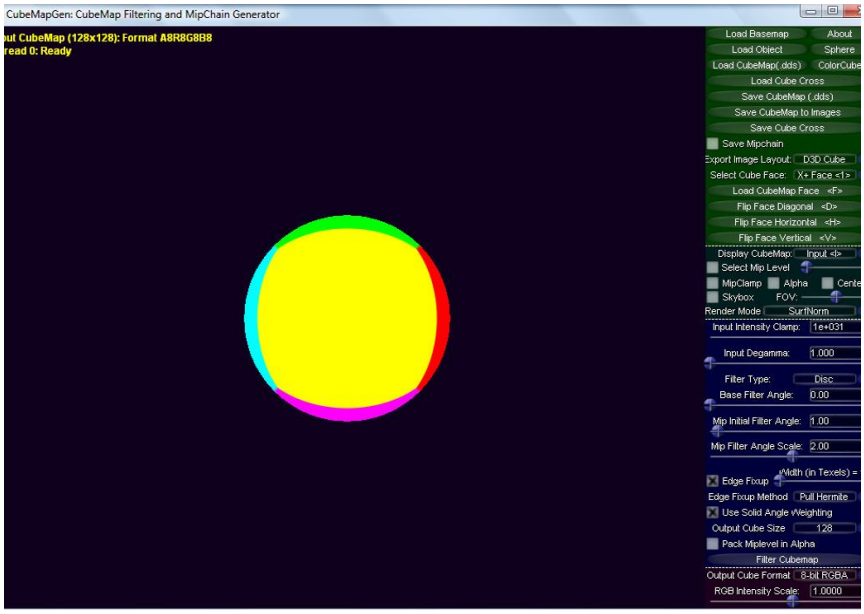


**Figura 3.8:** Schermata iniziale di CubeMapGen

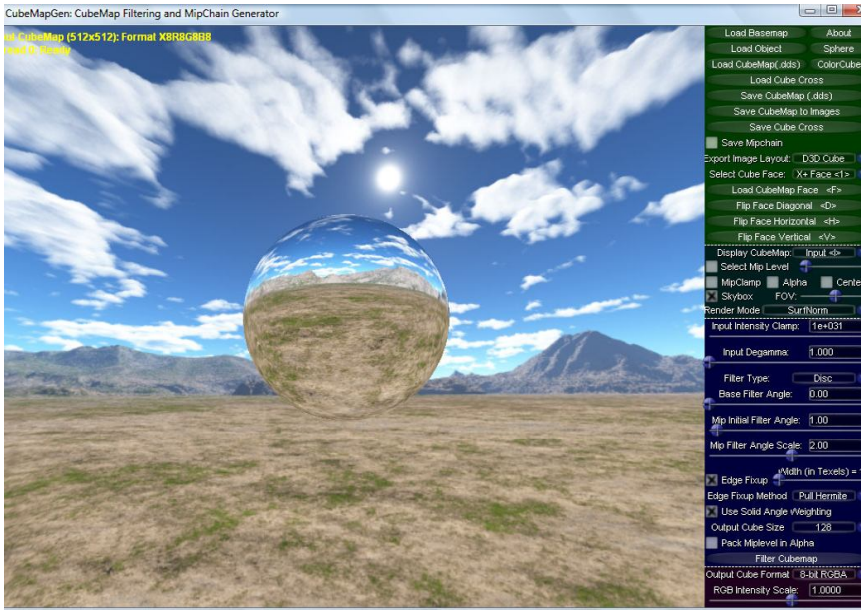


**Figura 3.9:** CubemapGen: anteprima dello skybox

# Capitolo 4

# Descrizione del sistema

## 4.1 Descrizione generale

Il progetto realizzato consiste in un simulatore tridimensionale di gruppi di persone in movimento (figura 4.1). All'avvio dell'applicazione viene visualizzato uno scenario (per il momento si tratta di un prato e di alcuni elementi architettonici) nel quale si muovono un certo numero di persone suddivise per gruppi. Il numero di gruppi, la loro dimensione e la traiettoria seguita sono parametrizzabili. Il tutto è strutturato in modo che le persone non possano collidere nè tra gli appartenenti al loro stesso gruppo nè con persone di altri gruppi: nel caso due pedoni vengano ad essere troppi vicini, ognuno di essi farà in modo di spostarsi per evitare lo scontro con l'altro. L'obbiettivo è quello di ottenere una simulazione real-time nella quale i comportamenti siano il più possibile realistici.

E' inoltre possibile intervenire a runtime sulla simulazione (attraverso i tasti speciali di tabella 4.1) per modificarne o analizzarne alcune caratteristiche particolari. Si può ad esempio aumentare o diminuire la velocità di spostamento dei pedoni, identificare (attraverso cerchi di colori diversi posti sul terreno) a quale gruppo appartiene ciascun pedone, visualizzare il movimento dello scheletro interno delle persone, entrare in modalità wireframe e ricavare alcuni dati numerici sull'andamento della simulazione.

L'obbiettivo è stato quello di realizzare un software che possa un giorno essere impiegato per simulare il movimento dei gruppi di persone rilevate da un robot.

**Figura 4.1:** Screenshot del simulatore realizzato

| **Tasti speciali** | **Funzione** |
|---|---|
| SPACE | Pietrifica i pedoni immobilizzandoli |
| F5 | Cambia la velocità di spostamento dei pedoni |
| F6 | Visualizza il gruppo di appartenenza di ciascun pedone |
| F7 | Visualizza il movimento degli scheletri |
| F8 | Modalità wireframe |
| F9 | Statistiche (FPS, tempo di CPU, occupazione della memoria) |

**Tabella 4.1:** Tasti speciali e relative funzioni

È infatti in progetto l'idea di sviluppare, all'interno dei laboratori dell'università, un robot in grado, tramite un laser, di individuare dei gruppi di persone in movimento all'interno di una stanza. In futuro si pensa quindi di memorizzare sia la dislocazione dei muri, sia le posizioni in cui i gruppi si vengono a trovare istante per istante, e di riprodurre poi tutto questo al calcolatore tramite il simulatore.

## 4.2 Sviluppo del sistema

La prima fase del lavoro svolto è stata dedicata al reperimento e allo studio di una documentazione il più possibile ampia sulla simulazione di folle, al fine di inquadrare il problema e di mettere a fuoco quali fossero gli aspetti che si sarebbero poi dovuti prendere in considerazione nelle fasi seguenti, nonchè farsi un'idea dei possibili strumenti e algoritmi da utilizzare.Il secondo passo è stato installare Horde3D, studiandone le caratteristiche e il funzionamento al fine di riuscire a padroneggiare il meglio possibile lo strumento.

Durante questa fase preliminare è stato in particolare preso in considerazione un esempio già presente all'interno dell'SDK (*Sample Chicago*, figura 4.2). Questo semplice programma permette di visualizzare un unico gruppo (di 100 persone) che camminano senza urtarsi ma in modo casuale all'interno di in un cerchio di dimensioni definite. Il gruppo nel suo complesso resta sempre fermo nella stessa posizione al centro della scena, mentre i modelli umani erano tutti uguali e presentavamo movimenti limitati (spostano solamente un po' più in alto o un po' più in basso i piedi, lasciando la gamba rigida). Anche lo scenario è piuttosto scarno i pedoni camminano su un piano sospeso a mezz'aria, mentre sullo sfondo si intravede un tramonto. Lo studio di questo esempio ci ha permesso di capire se Horde3D fornisse realmente tutti gli strumenti adatti a realizzare il simulatore che era in progetto di creare, ed ha costituito una prima base di partenza per la realizzazione simulatore stesso.

La fase di lavoro successiva è consistita nella creazione di nuovi modelli animati di persone. A questo scopo si sono valutati un buon numero di software di modellazione 3D, fino a trovarne uno che consentisse di raggiungere tale obbiettivo. La difficoltà maggiore è stata individuare uno strumento che permettesse sia di creare da zero (o di reperire con facilità in rete) mesh di donne e uomini, sia di animarle, sia di importarle in modo corretto in un formato Collada compatibile con Horde3D. Molti dei programmi analizzati, infatti, consentivano di centrare un obbiettivo o due di quelli citati, ma non tutti e tre.

Superato questo problema, ci si è occupati dello scenario. Una volta capito che Horde3D si serviva di cubemap per introdurre immagini di sfondo nelle scene, si è andati alla ricerca di un software in grado di crearne una adatta al motore grafico


**Figura 4.2:** Sample Chicago

in esame. Infine si è ricorsi a GoogleSketchup per reperire ed esportare in formato Collada alcuni modelli di case, al fine di rendere il risultato finale più realistico e piacevole da vedere (vedi figura 4.3).

Parte cruciale del lavoro è stata quella dedicata allo sviluppo del codice vero e proprio. Partendo dall'esempio di cui si è parlato poco fa, si è inizialmente proceduto ad una prima opera di refactoring del codice, al fine di renderlo più leggibile ed efficiente. In seguito si è fatto in modo che i gruppi visualizzati non fossero più uno solo di dimensione fissa, ma potessero essere in numero arbitrario e di dimensione variabile. Si è quindi messo a punto un sistema per consentire a questi ultimi di muoversi secondo differenti traiettorie all'interno dello scenario. Poi sono stati inseriti controlli per fare in modo che i pedoni controllassero di non scontrarsi non solo con gli appartenenti al loro stesso gruppo, ma anche con tutte le persone dei gruppi adiacenti. Fatto questo, sono state aggiunte le funzioni collegate ai tasti speciali in tabella 4.1.

A conclusione di tutto si è infine fatta una analisi delle prestazioni del software realizzato.


**Figura 4.3:** Sample Chicago

## 4.3 Animazione dei personaggi

I personaggi umani che popolano il progetto sono stati realizzati a partire da mesh rigide preesitenti trovate in rete. Per creare l'animazione di movimento è necessario importare tali mesh in 3DS Max, collocandole al centro delle coordinate spaziali, con i piedi appoggiati al piano xy (figura 4.4).

Bisogna poi andare su **Create->System** e selezionare **Biped**: cliccando e trascinando nella vista prospettica si crea uno scheletro la cui altezza dovrà essere il più possibile simile a quella della mesh scelta. Il passo successivo consiste nel mettere opportunamente in posa lo scheletro, in modo da farlo combaciare il più possibile con la mesh. A questo scopo è consigliabile selezionare preventivamente la mesh e, con un click del tasto destro, attivare prima **Object Properties -> See-Through** (per poter vedere lo scheletro attraverso la mesh) e poi **Freeze Selection** (per "bloccare" la mesh in modo che non possa più essere modificata).

Tutte le modifiche al biped possibili vengono fatte attraverso il pannello **Motion**. Fatto questo, dopo aver selezionato un qualsiasi osso, si deve entrare in modalità **Figure Mode**. E' ora necessario spostare, ruotare e scalare le ossa (figura 4.6) fin quando non si raggiungerà un risultato soddisfacente. Per spostare le ossa simmet-

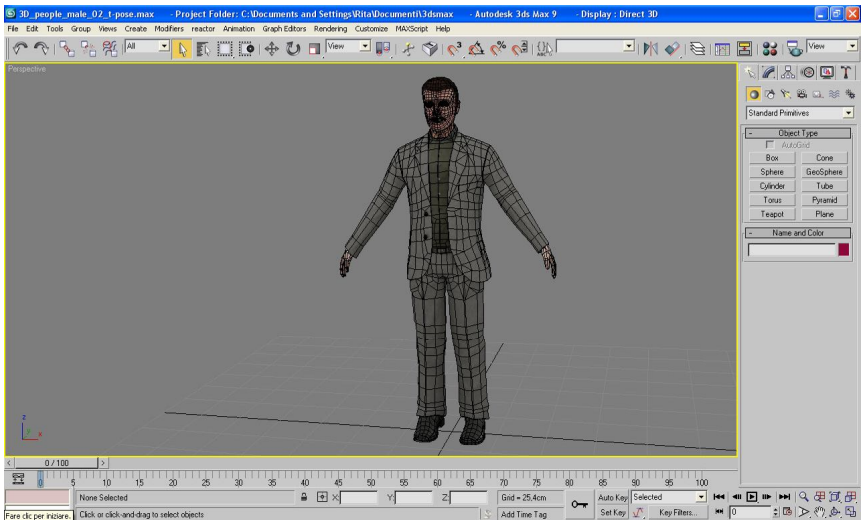

**Figura 4.4:** Mesh umana collocata al centro delle coordinate spaziali

ricamente è utile ricorrere alla funzione **Symmetrical**. Oltre a traslare le ossa, è consentito anche modificarne le dimensioni, variandone a piacimento l'ingombro.

Infine, in uscita dalla modalità **Figure Mode**, si procede con lo skinning vero e proprio, ossia procedere si associano i vari vertici della mesh alle diverse ossa dello scheletro, in modo che i primi seguano qualsiasi movimento del secondo. La prima cosa da fare è cliccare con il tasto destro sulla mesh e selezionare **Unfreeze All**. A questo punto, con la mesh selezionata, si va sul menu **Modify** e nella **Modifier List** si seleziona **Skin**. Ora si clicca su **Add** e si selezionano i nomi di tutte le ossa del biped (caratterizzate dal prefisso bip01).

A questo punto, si entra in modalità **Edit Envelopes**, per editare l'inviluppo e la pesatura dei singoli vertici, ossia per specificare l'area di influenza di ciascun osso e, in particolare, il peso che ogni movimento dello stesso avrà sui vertici della mesh che lo circondano. Gli inviluppi sono rappresentati da maniglie collegate da sottili linee rosse (figura 4.8); per visualizzarli è sufficiente cliccare di volta in volta su ciascun osso dello scheletro. La loro forma ricorda quella di due capsule concentriche: quella più interna racchiude i vertici sui quali il movimento di quell'osso avrà un peso maggiore, quella esterna i vertici sui quali l'influenza dell'osso sarà minore. Il peso esercitato su ciascun vertice è indicato dai colori: i vertici su cui esso

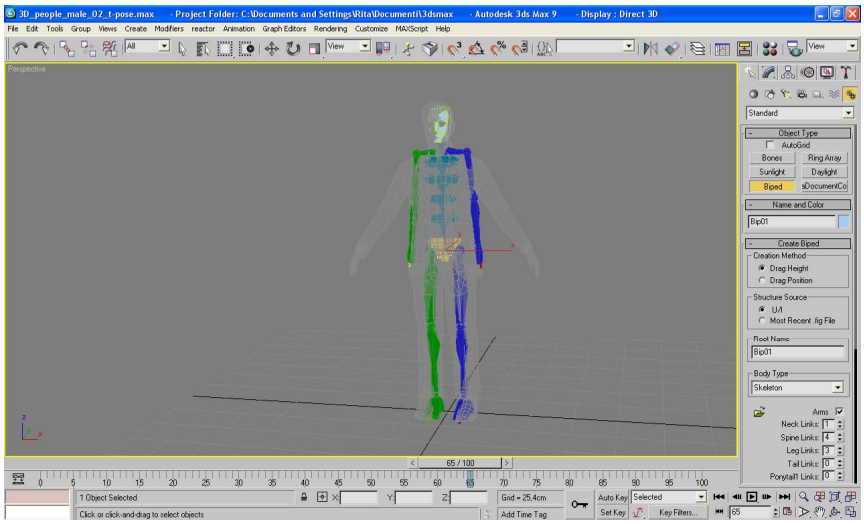

**Figura 4.5:** Mesh attraverso la quale si intravede lo scheletro

è maggiore appariranno rossi, quelli meno condizionati saranno blu. Per modificare gli inviluppi basta intervenire in modo opportuno sulle maniglie. Ai fini del successivo processo di animazione è fondamentale che ogni vertice sia sotto l'influenza di almeno un osso, altrimenti si otterrà uno sgradevole effetto di deformazione della figura.

L'ultimo passo del processo di creazione consiste nella sintesi dell'animazione vera e propria. A questo scopo, è necessario servirsi della barra di navigazione temporale nella parte bassa della finestra. La barra grigia con tacchette che si vede indica il susseguirsi del tempo: ogni quadratino rappresenta un frame. Il framerate può essere modificato tramite il pulsante **Time Configuration**. La prima cosa da fare è entrare in modalità **Auto Key**: da adesso in poi ogni volta che ci si posiziona su un determinato fotogramma e si sposta un certo osso dello scheletro, il fotogramma che si sta considerando cambia di colore, ad indicare che si tratta di un fotogramma chiave. Ciò significa che nel momento in cui si farà partire l'animazione (con gli appositi i pulsanti sulla destra) 3DS Max creerà l'animazione stessa andando ad interpolare le posizioni assunte dallo scheletro nei diversi fotogrammi chiave. È necessario quindi spostarsi su fotogrammi opportunamente distanziati e traslare e/o ruotare in modo adeguato i giunti dello scheletro fino ad ottenere un'animazione di

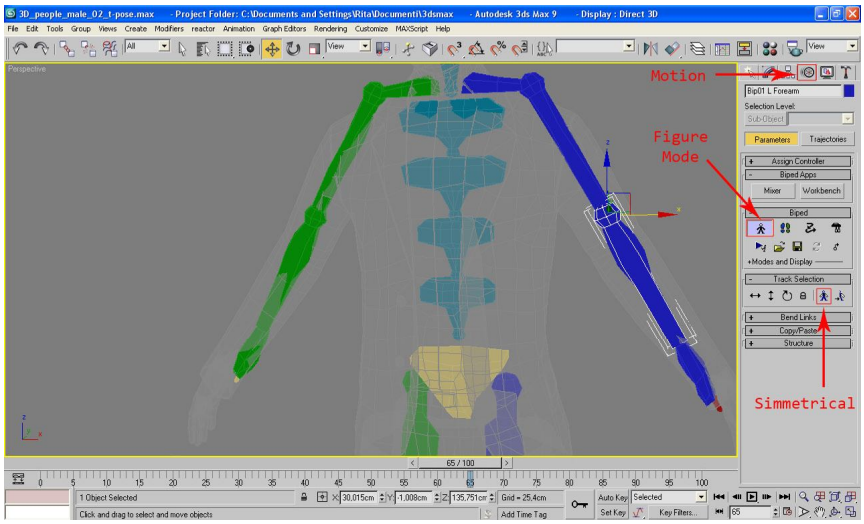


**Figura 4.6:** Adattamento dello scheletro alla mesh

camminata il più possibile realistica.

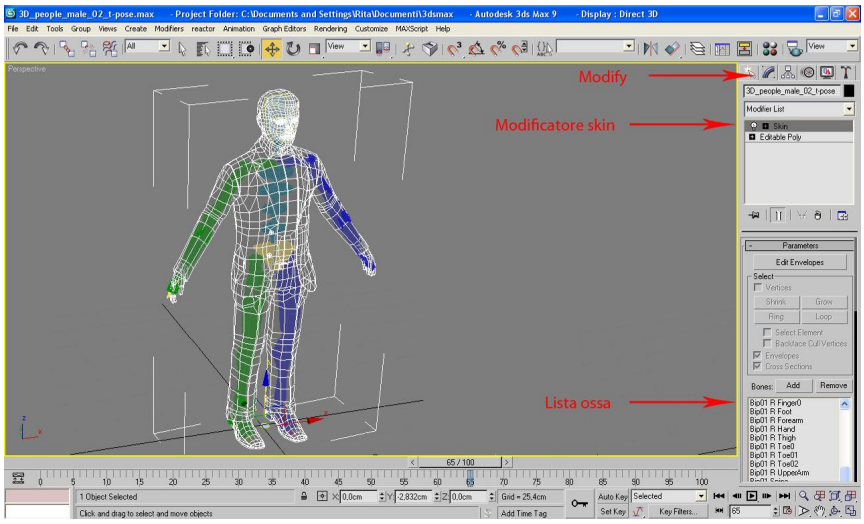


**Figura 4.7:** Modificatore skin

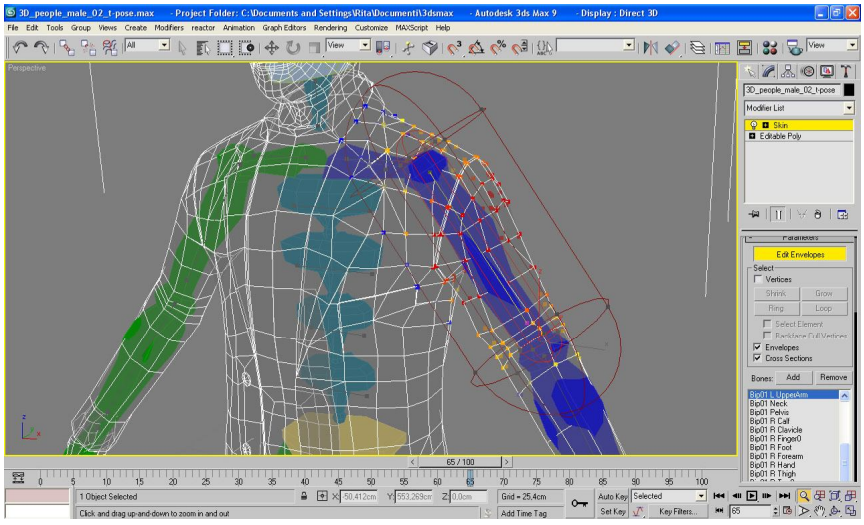

**Figura 4.8:** Involucri e loro influenza sui vertici

## 4.4 Struttura del codice

Il software sviluppato si articola su un main file e cinque classi (figura 4.9):

- Point
- Person
- Group
- Crowd
- Application

Il **main** si occupa di effettuare tutte le opportune chiamate a funzioni di GLFW[1] per creare la finestra del programma e predisporre listener per la pressione di tasti della testiera e per i movimenti del mouse. Inoltre, sempre nel programma principale viene impostato il frame rate e viene invocato ad ogni frame il ciclo di rendering dell'applicazione, il cui codice è contenuto nella classe Application.

La classe **Point** è una classe molto semplice, creata per memorizzare le coordinate di punti 2D in modo più leggibile.

La classe **Person** memorizza i dati inerenti ad una singola persona, quindi in particolare le coordinate della posizione attuale e della destinazione futura. Va inoltre detto che la posizione futura di ogni pedone è calcolata tenendo conto di un modello basato su forze. Si assume cioè che ogni pedone risenta di una forza attrattiva esercitata dal proprio punto destinazione e di una forza repulsiva esercita da ciascuna delle persone vicine (per maggiori dettagli, si rimanda al paragrafo 4.6.3). Il vettore forza totale agente sul pedone è calcolato come somma di queste componenti, le quali verranno anch'esse memorizzate dalla classe Person. Ovviamente, saranno presenti anche i necessari metodi per leggere i dati.

La classe **Group** archivia le informazioni relative al singolo gruppo, quindi il numero dei suoi appartenenti, le coordinate del baricentro [2], le coordinate del centro del cerchio all'interno del quale verranno scelte le destinazioni future di ciascuna

[1] GLFW è una libreria gratuita, Open Source e multi-piattaforma per creare contesti OpenGL e gestire il tempo e l'input, sia esso da tastiera, mouse o joystick.

[2] calcolato come punto le cui coordinate rappresentano il valor medio delle ascisse e delle ordinate dei punti in cui si trovano tutti i componenti del gruppo

persona (vedi paragrafo4.6), ecc. Tra i suoi metodi riveste particolare importanza *init()*, che si occupa di assegnare ad ogni membro del gruppo un template umano, il modello di un cerchio di un determinato colore (uguale per tutto il gruppo, potrà essere utilizzato per distinguere tra di loro a runtime gli appartenenti ai diversi gruppi) e un'animazione di camminata.

La classe **Crowd** non è altro che un'estensione della classe *vector* della Standard Template Library pensata per contenere puntatori ad oggetti di tipo Group. Ai metodi della classe vector ne viene aggiunto solo uno (*calGroupDistance()*), che si occupa di calcolare le distanze tra i baricentri dei vari gruppi e, nel caso alcune di esse siano più piccole di una certa soglia, di fare in modo che siano effettuati opportuni controlli per evitare collisioni tra membri di gruppi vicini.

La classe **Application** è il vero nucleo del software. Tale classe si incarica di gestire, attraverso i suoi metodi, il ridimensionamento della finestra, la pressione dei tasti, il movimento del mouse, ecc. Il metodo *init()* è si occupa di caricare tutte le risorse grafiche che andranno a comporre lo scenario, di creare una telecamera, di definire la sorgente di luce e di istanziare i vari gruppi che parteciperanno alla simulazione. Inoltre il metodo *mainLoop()* rappresenta, come suggerisce il nome stesso, il mainloop dell'applicazione: si tratta cioè di una funzione che viene invocata ad ogni frame e che ha il compito di fare in modo che vengano aggiornate le animazioni; prima del suo termine viene poi invocata *h3dRender()*, che insieme ad altre funzioni dell'engine ordina a quest'ultimo di effetturare il rendering vero e proprio.

Quindi, volendo riassumere, il simulatore nel suo complesso è costituito da un programma principale che definisce le proprietà più generali del software, per poi andare a richiamare i metodi di un'instanza di Application sia al momento dell'inizializzazione sia durante tutto il corso della simulazione stessa, per aggiornare le animazioni e le posizioni delle persone. La classe Application, oltre a svolgere le funzioni viste prima, istanzierà un oggetto Crowd, creando di fatto un vettore di puntatori a Group ad ognuno dei quali verranno assegnati certi parametri (e, soprattutto, determinati punti di controllo attraverso i quali dover passare). Ciascun oggetto Group contiene infine un insieme di n Person. Sia la classe Application sia la classe Group si servono poi della classe Point per memorizzare le coordinate di alcuni punti 2D.

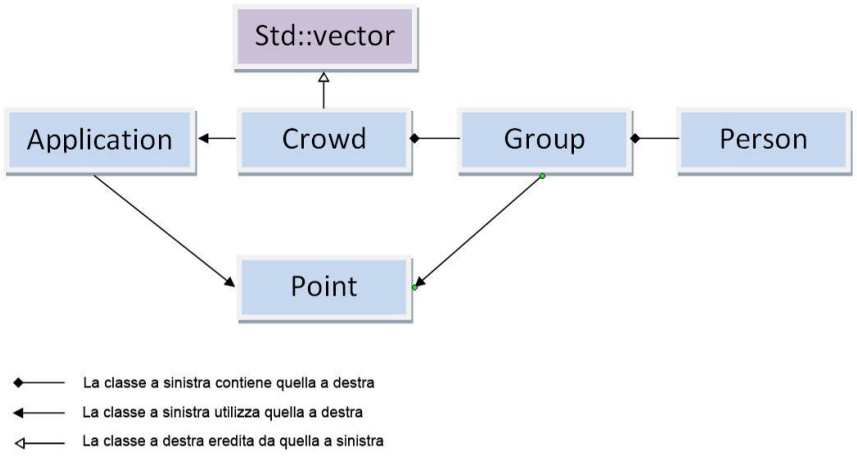


**Figura 4.9:** Classi che compongono l'applicazione.

## 4.5 Creazione dello scenario

Lo scenario del simulatore è costituito da un piano erboso sul quale sorgono alcuni edifici ed una fontana (figura 4.10).Tali elementi sono stati realizzati ed esportati in formato Collada con Google Sketchup Pro (vedi capitolo 3 sezione 3.3). Una volta posizionati i relativi file nelle cartelle *animations* e *models*, essi sono richiamati all'interno del programma dalla classe *Application* (listato 4.1). E' infatti il metodo *init()* di tale classe ad occuparsi di includere nell'applicazione tutte risorse grafiche esterne necessarie. Sempre da qui è possibile applicare operazioni di trasformazione ai modelli, ruotandoli, traslansoli e/o scalandoli.

Attraverso un procedimento analogo si inseriscono anche i modelli delle persone e dei cerchi colorati che si possono visualizzare sotto ciascun pedone (tasto F6) per identificarne il gruppo di appartenenza. L'unica differenza è che, in questo caso, è necessario associare a ogni modello umano anche la relativa animazione. Poichè a seconda del gruppo di cui ogni pedone fa parte dovrà essere assegnato a quest'ultimo un cerchio di un colore ben preciso, si è scelto di fare in modo che tutte queste operazioni siano compiute dal metodo *init()* della classe *Group*.

Lo sfondo dello scenario non è altro che una cubemap creata con CubeMap-

**Figura 4.10:** Classi che compongono l'applicazione.

```
H3DRes house1Res=h3dAddResource(H3DResTypes::SceneGraph,
                    "models/house1/house1.scene.xml", 0 );
h3dutLoadResourcesFromDisk( _contentDir.c_str() );
H3DNode house1 = h3dAddNodes( H3DRootNode, house1Res );
h3dSetNodeTransform( house1, 60, 0, 0, 0, 0, 0,
                              0.23f, 0.23f, 0.23f );
```

**Listato 4.1:** Esempio di codice necessario ad inserire nello scenario il modello di un edificio

Gen (vedi capitolo 3 sezione 3.5). Una volta ottenuto il file .dss è necessario collocarlo nella cartella *Horde3D\Binaries\Content\models\skybox* ed editare il file *skybox.material.xml* (listato 4.2), assegnando il percorso del file creato come valore dell'attributo *map*. Il file *skybox.scene.xml* sarà poi referenziato nel codice dell'applicazione a partire dalla classe *Application*.

Il fatto che tutta la scena sia racchiusa all'interno di una struttura cubica spiega l'effetto di taglio (figura 4.11) che si osserva se si sposta troppo in alto il punto di vista: è come se l'osservatore si allontanasse così tanto dall'origine degli assi da uscire dal cubo all'interno del quale lo scenario è contenuto.

```
<Material>
        <Shader source="shaders/skybox.shader"/>
        <Sampler name="albedoMap"
                 map="models/skybox/sky.dds" />
</Material>
```

**Listato 4.2:** Contenuto del file skybox.material.xml

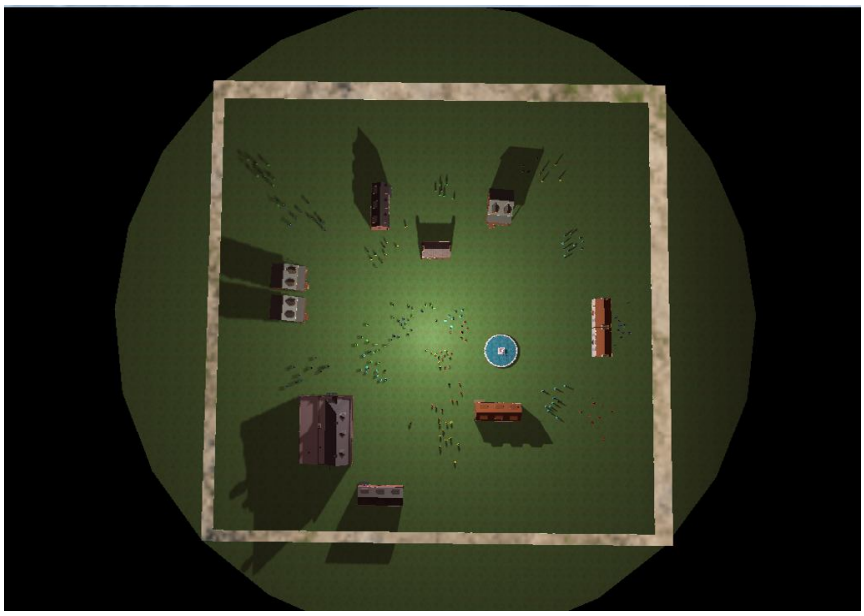

**Figura 4.11:** La scena vista a grande distanza dall'origine degli assi

Infine, sempre nella classe *Application* si trova il codice necessario a creare e posizionare la sorgente di luce della scena. Di essa è possibile definire, oltre alla collocazione, anche il colore, il raggio, il Fov (Field Of View), ecc.

```
H3DNode light = h3dAddLightNode( H3DRootNode, "Light1",
                lightMatRes, "LIGHTING", "SHADOWMAP" );
h3dSetNodeTransform(light, -20, 80, 0, -70, 0, 0,
                           1, 1, 1 );
h3dSetNodeParamF(light, H3DLight::RadiusF, 0, 1500 );
h3dSetNodeParamF(light, H3DLight::FovF, 0, 200 );
h3dSetNodeParamI(light, H3DLight::ShadowMapCountI, 3 );
h3dSetNodeParamF(light, H3DLight::ShadowSplitLambdaF,
                           0, 0.9f);
h3dSetNodeParamF(light, H3DLight::ShadowMapBiasF,
                           0, 0.001f);
h3dSetNodeParamF(light, H3DLight::ColorF3, 0, 0.9f );
h3dSetNodeParamF(light, H3DLight::ColorF3, 1, 0.9f );
h3dSetNodeParamF(light, H3DLight::ColorF3, 2, 0.9f );
```

**Listato 4.3:** Creazione della sorgente di luce della scena

## 4.6 Movimento dei gruppi

Vediamo ora come è stata ottenuta la navigazione dei gruppi all'interno della scena.

### 4.6.1 Schieramento iniziale

Per prima cosa quando si va ad istanziare un gruppo questo dovrà essere inizializzato tramite il metodo *Group::init()*. Tale metodo richiede, come parametri in ingresso, il numero **pNum** di componenti del gruppo, le coordinate del punto in cui si desidera che il gruppo si trovi alla partenza (che indicheremo con pInizx e pInizz) ed il colore che si vuole associare al gruppo per distinguerlo dagli altri [3]. Va ricordato che per ogni gruppo verranno istante per istante memorizzati il baricentro **B(bx,by)** e un punto **C(cx,cy)** che chiameremo "centro delle destinazioni". Alla partenza, baricentro e centro delle destinazioni coincidono con il punto iniziale. Ogni gruppo parte in formazione (figura 4.12) vale a dire che tutti i suoi appartenenti si dispongono lungo una circonferenza centrata nel punto iniziale di raggio

$$radius = pNum * 2 \tag{4.1}$$

[3] sono attualmente possibili cinque colori: blu, giallo, verde, rosso e azzurro

Per la j-esima persona del gruppo le coordinate del punto px e pz in cui essa si troverà all'avvio della simulazione sono calcolate come

$$px_j = sin\left(\frac{j}{pNum} * 2\pi\right) * radius + pInizx \tag{4.2}$$

$$pz_j = cos\left(\frac{j}{pNum} * 2\pi\right) * radius + pInizz \tag{4.3}$$

dove $j$ è ovviamente compreso tra 1 e pNum. Applicando questa formula per ogni persona, il risultato sono proprio una serie di punti disposti su una circonferenza di raggio $radius$ centrata in $(pInizx, pInizz)$.

**Figura 4.12:** Disposizione iniziale dei componenti di un generico gruppo.

### 4.6.2 Scelta della destinazione futura

La scelta della destinazione futura di ciascuna persona avviene ad opera del metodo *Group::chooseDestination()*, il cui codice è riportato nel listato 4.4:

Si osserva facilmente che per ciascun pedone il punto **D(dx,dz)** verso cui dirigersi è scelto casualmente all'interno di un cerchio di raggio $radius$ centrato in C(cx,xz), dove C è quello che abbiamo prededentemente chiamato "centro delle destinazioni".

```
void Group::chooseDestination( Person &p )
{
        float ang = ((rand() \% 360) / 360.0f) * 6.28f;
        float rad = (rand() \% radius);

        float dx, dz;
        dx = sinf( ang ) * rad + cx;
        dz = cosf( ang ) * rad + cz;
        p.setDestination(dx,dz);
}
```

**Listato 4.4:** Codice del metodo *Group::chooseDestination()*.

Prima viene scelto un angolo casuale fra 0 e $2\pi$ , in modo da fissare la direzione nella quale D dovrà trovarsi; dopodichè le coordinate di D vengono calcolate in modo che il punto venga a trovarsi ad una distanza da C compresa tra 0 e $radius$ (vedi figur 4.13).

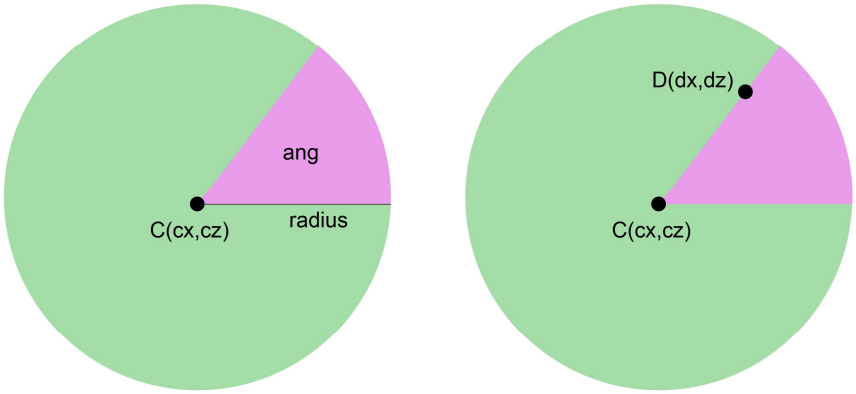


**Figura 4.13:** Funzionamento del metodo *Group::chooseDestination()*.

Ma dove si trova di preciso il punto C? Se il suo valore fosse costante il gruppo nel suo complesso rimarrebbe sempre fermo nella stessa zona, perché i suoi compo-

nenti tenterebbero di dirigersi verso punti tutti collocati all'interno dello stesso cerchio. Per fare in modo che i gruppi si spostino, basta variare le coordinate del punto C, cosicchè (il modo in cui questo avviene sarà spiegato nel dettaglio più avanti in questa sezione) raggiunta la loro meta le persone se ne prefiggano un'altra in un cerchio collocato in un'altra parte della scena; per raggiungere tale destinazione i pedoni saranno costretti a spostarsi, iniziando così la loro navigazione.

Assegnando opportuni valori alle coordinate di C si può quindi stabilire il percorso seguito dal gruppo. Tali valori sono attualmente definibili dal programmatore tramite codice al momento della creazione del gruppo, come da listato 4.5. *addPoint()* è un metodo della classe Group che inserisce i punti passatigli come argomento in un vettore di oggetti Point che conterrà la sequenza di checkpoint attraverso i quali il gruppo dovrà passare.

```
_newGroup = new Group( _contentDir );
_newGroup->init(10, Point(60,70), AZZURRO);
_newGroup->addPoint(Point(70,130) );
_newGroup->addPoint(Point(10,130) );
_newGroup->addPoint(Point(10,60) );
_newGroup->addPoint(Point(110,10) );
_newGroup->addPoint(Point(180,0) );
_groups.push_back(_newGroup);
```

**Listato 4.5:** Creazione di un nuovo gruppo.

Il codice del programma è scritto in modo che il valore del centro delle destinazioni venga aggiornato solo quando tutti i componenti del gruppo sono arrivati ad una distanza del centro delle destinazioni corrente pari almeno a pNum*2. Questo controllo serve ad evitare situazioni di dispersione come quella mostrata in figura 4.14. Vediamo qui un gruppo di sei elementi (in nero) che deve raggiungere in successione tre checkpoint C1, C2, C3 (in rosso). In giallo sono evidenziati dei cerchi all'interno dei quali possiamo ragionevolmente pensare che la destinazione sia considerata raggiunta (si rammenti infatti che le destinazioni vengono scelte all'interno di un cerchio centrato in C di raggio pNum*2). Osserviamo cosa potrebbe accadere senza la restrizione appena descritta:

- All'istante (a) il centro delle destinazioni attivo è C1: tutti i componenti del gruppo cercano di raggiungere una destinazione nel cerchio giallo ad esso circostante;
- All'istante (b) un pedone (in verde) raggiunge la propria destinazione: il nuovo centro delle destinazioni diventa C2, per cui il pedone inizia a dirigersi verso quella direzione.
- All'istante (c) alcuni pedoni (in rosa) hanno raggiunto la loro prima destinazione e si avviano verso il secondo punto di controllo, ma contemporaneamente la persona in verde è arrivata nuovamente a destinazione, attivando C2;
- All'istante (d) i pedoni rosa, dopo aver raggiunto la loro seconda destinazione, ne scelgono correttamente una terza nell'intorno di C3; il problema è che le persone in blu si erano per qualche motivo attardate (probabilmente rallentate dalla presenza di ostacoli lungo il cammino), per cui nell'istante in cui raggiungono la loro prima destinazione, C3 si è già attivato, perciò esse si avviano verso di esso, evitando quindi C2.

A tutto questo si può ovviare se, come appena descritto, si aspetta che tutti gli appartenenti al gruppo siano giunti nei pressi della propria meta prima di modificare le coordinate del centro delle destinazioni. Nell'attesa, i pedoni che sono arrivati per primi si limiteranno a "rimanere in zona" scegliendosi un'altra destinazione sempre nell'intorno del centro delle destinazioni attuale. Il risultato ottenuto è quindi corretto ma anche più realistico, perché è verosimile aspettarsi che anche nella realtà gli appartenenti di uno stesso gruppo si attendano piuttosto che proseguire per la loro strada dimenticandosi dei ritardatari.

### 4.6.3 Collision avoidance

Per capire come viene realizzata la collision avoidance tra le persone che prendono parte alla simulazione, ricordiamo innanzitutto che si sta considerando ogni pedone come un oggetto puntiforme. Attorno ad esso si immagina di avere tre zone repulsive, ad ognuna delle quali viene associato un certo coefficiente di repulsione.

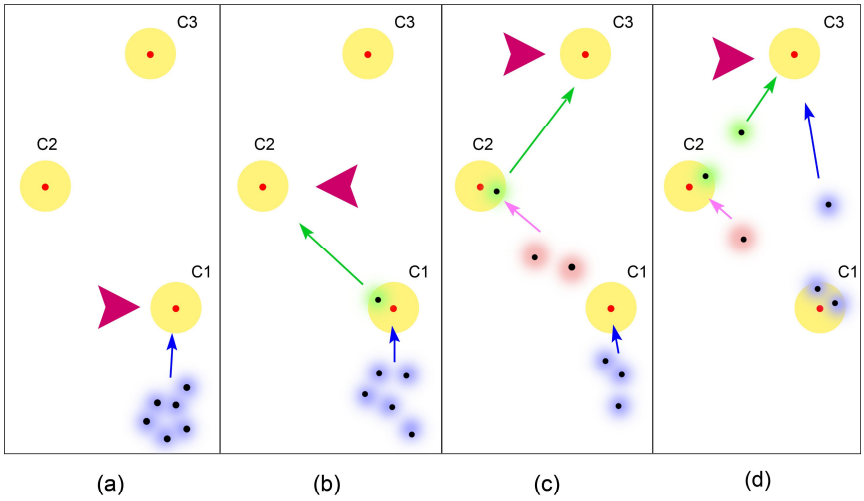


**Figura 4.14:** Esempio di dispersione del gruppo.

Osservando la figura 4.15 si può notare che

$$d1 < d2 < d3 \tag{4.4}$$

$$f1 > f2 > f3 \tag{4.5}$$

ossia minore è la distanza dal punto in cui si trova la persona p in esame (centro dei cerchi concentrici), maggiore è il coefficiente repulsivo. Tali coefficienti rappresentano di fatto il modulo della forza repulsiva avvertita da un qualsiasi altro pedone p2 che si trovi nelle vicinanze di p.

Ogni volta che si rende necessario calcolare la nuova posizione che p dovrà assumere al frame successivo i valori di posizione e orientamento correnti verranno aggiornati tenendo conto (nel modo appena illustrato) dalle forze repulsive esercitate su p dalle persone adiacenti e della forza attrattiva esercitata su p dal punto di destinazione D.

A calcolare la forza risultante è il metodo *Group::update()*. Per ogni membro del gruppo esso parte azzerando le componenti fx ed fz del vettore forza complessiva e

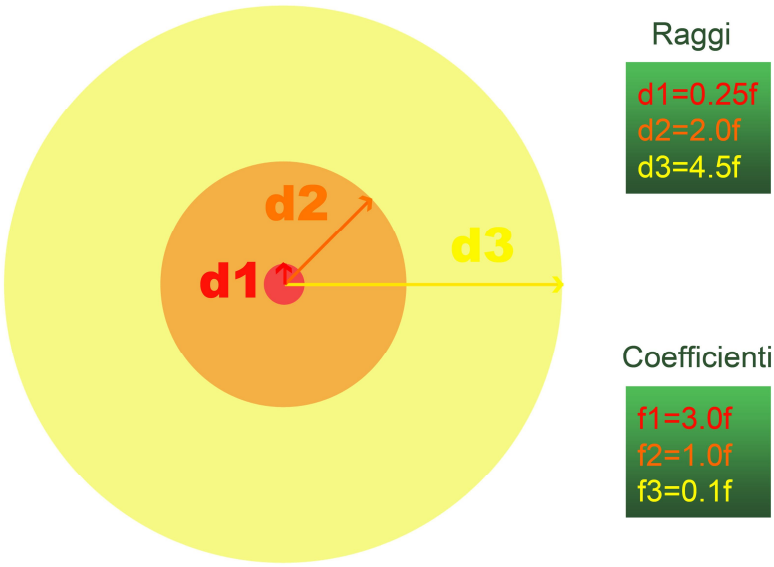


**Figura 4.15:** Zone di repulsione attorno a ciascun pedone.

calcolando il modulo della distanza della persona rispetto alla destinazione

$$dist = \sqrt{(dx - px)^2 + (dz - pz)^2} \tag{4.6}$$

Se $dist < 3.0$ la destinazione viene considerata raggiunta e ne viene calcolata un'altra; altrimenti vengono calcolate le componenti della forza attrattiva normalizzata verso destinazione in questo modo:

$$afx = \frac{dx - px}{dist} \tag{4.7}$$

$$afz = \frac{dz - pz}{dist} \tag{4.8}$$

e il risultato viene sommato alla forza totale secondo un opportuno coefficiente:

In modo analogo, vengono calcolate anche le forze repulsive tra p e tutte le altre persone appartenenti al medesimo gruppo (o ad altri gruppi sufficientemente

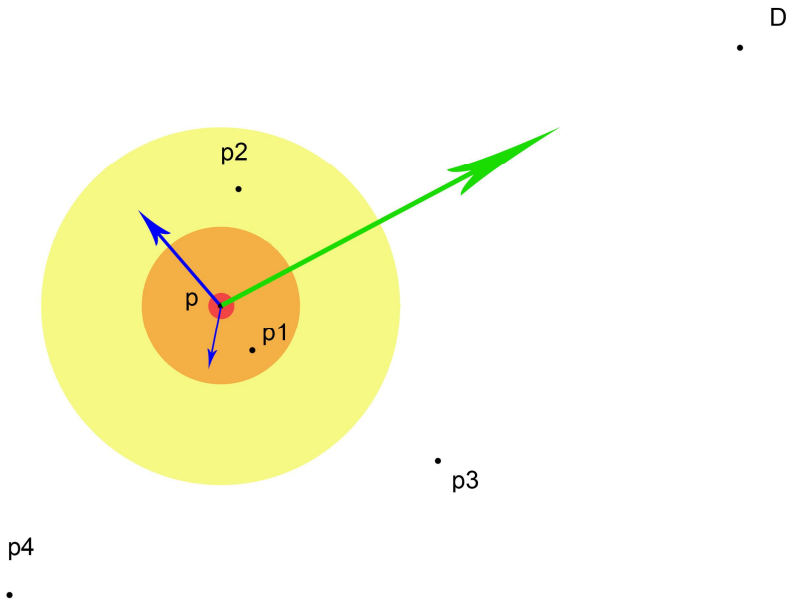


**Figura 4.16:** La forza totale agente sulla persona p sarà la somma vettoriale delle forze repulsive (in blu) e di quella attrattiva (in verde). Nota: il disegno non è in scala.

```
fx += afx * 0.035f;
fz += afz * 0.035f;
```

**Listato 4.6:** Le componenti della forza attrattiva vengono sommate a quelle della forza totale.

vicini)[4].

$$dist_2 = \sqrt{(px - px_2)^2 + (pz - pz_2)^2} \tag{4.9}$$

$$rfx = \frac{px - px_2}{dist_2} \tag{4.10}$$

$$rfz = \frac{pz - pz_2}{dist_2} \tag{4.11}$$

Ancora una volta, i valori ottenuti andranno sommati alle componenti della forza

[4] $dist_2$ è il modulo della distanza tra p e p2

totale moltiplicati per un certo coefficiente (vedi listato 4.7 ).

```
fx += rfx * strength;
fz += rfz * strength;
```

**Listato 4.7:** Le componenti della forza repulsiva vengono sommate a quelle della forza totale.

Però questa volta tale coefficiente, chiamato *strength*, non sarà sempre costante, ma varierà a seconda della distanza di p2 da p nel modo mostrato in listato 4.8.

```
if( dist2 <= d3 && dist2 > d2 )
{
        float m = (f3 - 0) / (d2 - d3);
        float t = 0 - m * d3;
        strength = m * dist2 + t;
}
else if( dist2 <= d2 && dist2 > d1 )
{
        float m = (f2 - f3) / (d1 - d2);
        float t = f3 - m * d2;
        strength = m * dist2 + t;
}
else if( dist2 <= d1 )
{
        float m = (f1 - f2) / (0 - d1);
        float t = f2 - m * d1;
        strength = m * dist2 + t;
}
```

**Listato 4.8:** Calcolo dei coefficienti per la forza repulsiva a seconda della distanza d dal pedone in esame. I valori d1, d2, d3, f1, f3, f4 sono quelli mostrati in figura 4.15

Per ridurre i tempi di calcolo, i coefficienti $m$ e $t$ non vengono ricalcolati ogni volta, ma vengono assegnati i seguenti valori costanti precalcolati

$$m1 = -8, \qquad m2 = -0.514286, \qquad m3 = -0.04 \tag{4.12}$$

$$t1 = 3, \qquad t2 = 1.12857, \qquad t3 = 0.18 \tag{4.13}$$

Il codice visto prima si riduce quindi a quello del listato 4.9.

```
if( dist2 <= d3 && dist2 > d2 )
        strength = m3 * dist2 + t3;
else if( dist2 <= d2 && dist2 > d1 )
        strength = m2 * dist2 + t2;
else if( dist2 <= d1 )
        strength = m3 * dist2 + t3;
```

**Listato 4.9:** Calcolo dei coefficienti per la forza repulsiva a seconda della distanza d dal pedone in esame: versione sintetica.

A questo punto è possibile finalmente aggiornare i valori del vettore posizione $\vec{P}(px, pz)$ e orientamento $\vec{O}(ox, oz)$ per il pedone in esame sulla base del valore delle componenti ottenute per il vettore forza $\vec{F}(fx, fz)$. Si può osservare come il nuovo orientamento (direzione) sia calcolato come semisomma tra il vettore orientamento corrente e il vettore forza totale.

$$\vec{P'} = \vec{P} + \vec{F} \tag{4.14}$$

$$\vec{O'} = \frac{\vec{O} + \vec{F}}{2} \tag{4.15}$$

### 4.6.4 Collision avoidance con membri di altri gruppi

Come visto, ad ogni frame viene fatto un controllo tra ogni pedone e tutti gli altri membri del suo gruppo per evitare che questi vadano in collisione. Ora, la cosa concettualmente più semplice sarebbe effettuare questo controllo con tutte le persone della folla anzichè solo con gli appartenenti al proprio gruppo. Questa soluzione, benchè praticabile, risulta computazionalmente troppo pesante. Un metodo più efficiente consiste dell'effettuare questo controllo solo con membri di gruppi talmente vicini da comportare il rischio di collisioni. Ad incaricarsi di capire se due gruppi

siano troppo vicini è il metodo *calcGroupDistance()* della classe Crowd, invocato ad ogni frame prima di fare l'update della posizione dei pedoni nei vari gruppi.

Il metodo *calcGroupDistance()* non fa altro che valutare la distanza relativa tra il baricentro Bi(bxi,bzi) di un gruppo e il baricentro Bj(bxj,bxj)[5] di ciascuno degli altri gruppi costituenti la folla. Se tale distanza risulta minore di un certo valore (di cui si parlerà nel paragrafo successivo), allora il vettore contenente tutti gli oggetti Person del secondo gruppo viene passato al primo gruppo, in modo che quest'ultimo abbia a disposizione le informazioni necessarie per calcolare le componenti repulsive di forza rispetto anche ai membri dell'altro gruppo in esame. Il vantaggio è che in questo modo vengono confrontate le posizioni solo delle persone ragionevolmente vicine tra di loro, mentre vengono ignorate quelle delle persone abbastanza lontane da non costituire un pericolo.

**Figura 4.17:** La presenza di cerchi colorati sotto ai pedoni permette di vedere come gli i diversi gruppi possano incrociarsi senza che ci siano collissioni tra i rispettivi membri.

Ma il concetto di "vicino" è molto relativo: quand'è che due gruppi sono sufficientemente vicini da costituire un pericolo l'uno per l'altro? Questo valore è fissato a MAX_ GROUP_ SIZE*4, dove MAX_ GROUP_ SIZE è una costante settata al

[5] $i = 1, ..., n \qquad j = 1, ..., n \qquad i \neq j$, dove n è il numero di gruppi che compongono la folla.

più grande numero di persone che si ipotizza possano entrare a far parte di un gruppo. Supponiamo infatti di avere MAX_ GROUP_ SIZE pari a dieci, e di considerare il caso in cui due gruppi di esattamente 10 persone (ossia per entrambi $pNum = 10$) si stiano avvicinando tra di loro. Siccome i membri di ciascun gruppo cercheranno una destinazione a distanza di massimo 2*pNum dal cerchio dal proprio centro delle destinazioni e tenderanno a viaggiare raggruppati a causa degli stratagemmi di cui si è parlato nel sottoparagrafo 4.6.2, non capiterà quasi mai di trovare una persona distante più di pNum*2 dal baricentro del proprio gruppo. Quindi se si effettua questo controllo di collisione quando i baricentri sono più vicini di pNum*4 (in questo caso 40) si è ragionevolmente sicuri di evitare qualsiasi tipo di scontro. Ciò si è verificato anche "sul campo" conducendo la simulazione con gruppi formati tutti da dieci persone e MAX_ GROUP_ SIZE settato a dieci: effettivamente, si osserva che le collisioni sono totalmente assenti.

Ovviamente, per essere più tranquilli si potrebbe aumentare questo valore, ma ciò significherebbe includere nei propri controlli un numero maggiore di gruppi e quindi di persone, aumentando notevolmente il carico computazionale rispetto al necessario.

Il caso appena analizzato può essere considerato come quello peggiore, perché se uno dei due o entrambi i gruppi in avvicinamento fossero di dimensione inferiore a quella massima fissata, allora, a maggior ragione, non ci sarebbe il rischio di eventuali collisioni. Infatti, più il gruppo è piccolo e più la quantità $pNum * 2$ è inferiore, per cui è più ristretta l'area, nelle vicinanze dei rispettivi baricentri, in cui è probabile trovare un qualche membro del gruppo stesso; pertanto, effettuando per ogni gruppo i controlli di non collizione a distanza MAX_ GROUP_ SIZE*4 dal rispettivo baricentro, siamo certi di non avere scontri. Assumendo di nuovo MAX_ GROUP_ SIZE*4=40 e considerando questa volta ad esempio un gruppo di due persone, allora $pNum * 2 = 4$, per cui è praticamente impossibile che qualche componente del gruppo venga a trovarsi a distanza dal suo baricentro maggiore di 40.

## 4.7 Analisi delle prestazioni

Dal grafico in figura 4.18 si può vedere come varia il numero di frame al secondo (FPS) aumentando il numero delle persone simulate. Le rilevazioni sono state fatte in due casi diversi:

- considerando i modelli "complessi" attualmente presenti nel simulatore, contenenti all'incirca 24800 triangoli cadauno (linea blu);
- considerando i modelli più "semplici" che erano disponibili con l'esempio di base scaricabile dal sito di Horde3D, costituiti da 972 triangoli e dotati di un'animazione molto più semplice, che coinvolge molti meno giunti (linea rossa).

Come paesaggio si è preso quello con prato e case che si può attualmente trovare nel simulatore, contenente nel suo complesso 32020 triangoli. Come si può notare, otte-

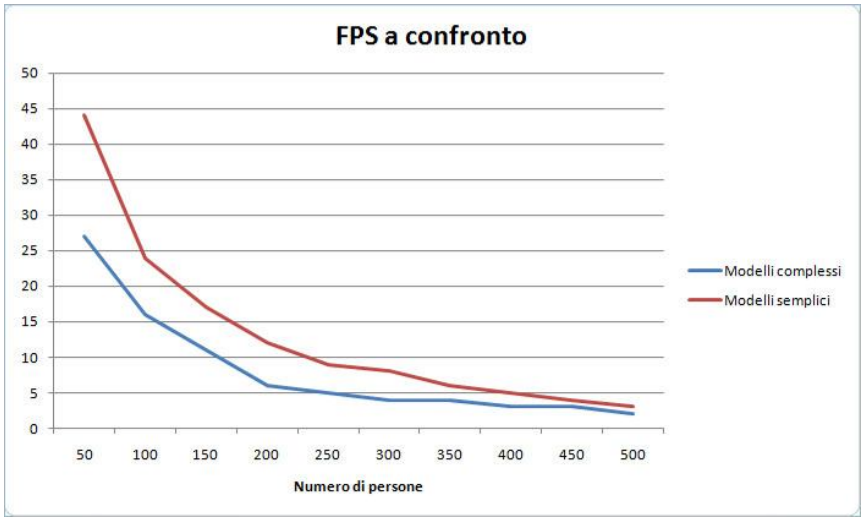


**Figura 4.18:** FPS al crescere del numero di persone simulate.

niamo un andamento iperbolico. Come era prevedibile, la situazione risulta migliore se ci si serve di modelli più semplici, riducendo così i tempi di calcolo per ciascun frame. Sperimentalmente si osserva che l'animazione è visivamente gradevole fino

circa agli 8 FPS, dopodichè inizia a diventare troppo scattosa. Prendendo questo valore come punto di riferimento, si vede che il massimo numero di pedoni che possono essere simulati in modo soddisfacente è di circa 300 con modelli semplici, 180 con quelli complessi. Lo svantaggio comportato dall'aumento di complessità è quindi tangibile. Ovviamente, nello scegliere quali modelli utilizzare bisognerà raggiungere un compromesso tra la gradevolezza visiva e il numero di pedoni che si vogliono avere.

Quindi per migliorare le prestazioni si può lavorare o sul codice (ottimizzandolo), o sul tipo di modelli utilizzati, o su una riduzione della complessità geometrica del paesaggio (anche se, nel caso in cui si usino modelli complessi, l'impatto del paesaggio rispetto alla complessità geometrica dei modelli risulta quasi trascurabile). Va inoltre aggiunto che i dati sono stati rilevati eseguendo la simulazione sotto Windows Vista con un notebook HP Pavillon dv6 dotato di processore Intel Core2 Duo T6500 @2.10GHz con 4GHz di RAM: è ragionevole aspettarsi che con un calcolatore più potente anche le prestazioni del simulatore tenderanno ad aumentare.

# Capitolo 5

# Conclusioni

Il lavoro svolto durante la realizzazione di questa testi ha riguardato inizialmente il reperimento e l'analisi di una documentazione il più esauriente possibile sul tema della simulazione di folle, per farsi una prima idea degli obbiettivi, delle problematiche e delle possibili soluzioni con cui chi si occupa di queste tematiche si trova a doversi confrontare.

La parte di sviluppo vera e propria ha invece riguardato principalmente i seguenti aspetti: la creazione di modelli umani tridimensionali animati, la realizzazione di uno scenario virtuale in cui far muovere la folla digitale e la realizzazione di un sistema software in grado di controllare il movimento dei gruppi di individui, facendo in modo che essi seguano specifici percorsi definiti dal programmatore come sequenza di punti e che non ci siano collisioni tra le persone.

Questo progetto presenta ancora molti margini di sviluppo e miglioramento:

- Nel più immediato futuro si pensa è di poterlo utilizzare congiuntamente al robot Pioneer, già presente nei laboratori del Dipartimento di Ingegneria dell'Informazione dell'Università di Parma. Questo robot, oggi in grado di riconoscere la presenza di persone tramite laser, potrà in futuro identificare anche interi gruppi di persone ed estrapolare informazioni sulla loro traiettoria a livello di gruppo. Le coordinate dei punti ricavate dal robot potranno essere inviate al simulatore, in modo che questo ricostruisca a calcolatore i gruppi rilevati con il laser e le rispettive traiettorie.

- Poiché il robot verrà utilizzato prevalentemente nei locali della facoltà, sarebbe anche interessante fare in modo che, date alcune informazioni come coordinate del punto medio, lunghezza e spessore dei muri, il simulatore fosse in grado di ricostruire in tempo reale la planimetria di un ambiente interno.

- La resa grafica sarebbe più realistica e gradevole con l'aggiunta di nuove animazioni per i modelli umani. Se ora essi si limitano a camminare, si potrebbe fare in modo che, in conseguenza a certi eventi, essi si mettano ad esempio a correre, a guardarsi intorno, a salutarsi, ecc.

- Attualmente, le persone che si trovano a dover aspettare altri membri del loro gruppo non si fermano, ma continuano a camminare con andatura costante nei dintorni del luogo in cui si trovano. Questo comportamento, già di per se' irrelistico, talvolta comporta che i pedoni in attesa compiano bruschi cambiamenti di direzione, ruotando su se stessi in modo anomalo. Ulteriori analisi andrebbero dunque condotte in questo senso per cercare di risolvere il problema.

- Altro risultato importante sarebbe ideare e implementare algoritmi di collision avoidance non solo tra un pedone e l'altro, ma anche tra le persone e gli elementi architettonici dell'ambiente circostante, funzionalità al momento totalmente assente.

- Infine, se si riuscissero a trovare o a estrapolare dati statistici sul comportamento delle folle in caso di particolari eventi come l'uscita dagli stadi, l'arrivo di una personalità famosa, ecc., li si potrebbe integrare nel simulatore in modo da sviluppare nel tempo un software utilizzabile dalle amministrazioni locali per prepararsi a fronteggiare a certe situazioni.

# Bibliografia


[1] http://legionsoftware.com/software/.

[2] Micheletti J. Varner D., Scott D. and Aicella G. Umsc small unit leader non-lethal trainer. *Proc. ITEC'98*, 2010.

[3] Dorigo M. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, 1992.

[4] Leandro Soriano Marcolino e Luiz Chaimowicz. Traffic control for swarm of robots: Avoid group conflicts. October 2009, IEEE/RSJ Internationa Conference on Intelligent Robots and Systems.

[5] http://www.massivesoftware.com/.

[6] Reynolds C. W. Flocks, herds and schools: a distributed behavioral model. 1987, SIGGRAPH Conference.

[7] Thalmann D. Musse S. R. A hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, April-June 2001.

[8] Thalmann D. Ulciny B. *Towards interactive real-time crowd behavior simulation*. December 2002.

[9] Overmars M. Kamphuis A. Finding paths for coherent groups using clearance. *Proceedings of the ACM SIGGHRAPH/Eurographics Symposium on Computer Animation*, 2004.

[10] Loscos C. Tecchia F. and Chrysanthou Y. Image-based crowd rendering. *IEEE Computer Graphics and Applications 22*, March-April 2002.

[11] Daniel Thalmann et al. Julien Pettré, Pablo de Haras Ciechomski. Real-time navigation crowds: scalable simulation and rendering. *Computer Animation and Virtual Worlds*, May 2006.

[12] Christopher Peters and Cathy Ennis (Trinity College Dublin). Modelling groups of plausible virtual pedestrian. *IEEE Computer Graphics and Applications*, July-August 2009.

[13] Cohen J et al. Luebke D., Reddy M. *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2002.

[14] http://www.horde3d.org/.

[15] http://www.ogre3d.org/.

[16] http://irrlicht.sourceforge.net/.

[17] https://collada.org/mediawiki/index.php/colladamax, 2010.

[18] http://www.w3.org/tr/rec-xml/.

[19] Justin A. Ehrlich and James R. Miller. A virtual environment for teaching social skills: Avisss. *IEEE Computer Graphics and Applications*, July-August 2009.

[20] Soraia Raupp Musse Daniel Thalmann. *Crowd Simulation*. Springer, 2007.

[21] N. Badler N. Pelachano, J. Allbeck. *Virtual Crowds: Methods, Simulation, and Control*. Morgan and Claypool Publishers, 2008.

[22] Stéphane Donikian Sébastien Paris. Activity-driven populance: A cognitive approach to crowd simulation. *IEEE Computer Graphics and Animation*, 2010.

[23] Seth Cooper Adrien Treuille and Zoran Popovic. Continuum crowds. 2006, International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH.

[24] Wolfram Burgard Boris Lau, Kai O. Arras. Multi-model hypothesis group tracking and group size estimation. *Proceedings of the IEEE ICRA 2009*, 2009.

[25] Julien Pettré Barbara Yersin and Daniel Thalmann. Crowd patches: Populating large-scale virtual environments for real-time applications. 2009, I3D Conference.

[26] Craig W. Reynolds. Steering behaviors for autonomous characters. 1999, Game Developers Conference.

[27] Yirgos Chrisanthou Alon Lerner, Eitan Fitusi and Daniel Cohen-Or. Fitting behaviours to pedestrian simulation. *Eurographics/ACM SIGGHRAPH Symposium on Computer Animation*, 2009.

[28] Paul Kanyuk (Pixar Animation Studios). Brain springs: Fast physics for large crowds in wall-e. *IEEE Computer Graphics and Applications*, July-August 2009.