Alma Mater Studiorum · Università di Bologna
Sede di Cesena
facoltà di scienze matematiche, fisiche e naturali
Corso di Laurea in Scienze e Tecnologie Informatiche

PROGETTAZIONE E REALIZZAZIONE DI UN SOFTWARE GRAFICO PER L'ASSEMBLAGGIO DI OGGETTI TRIDIMENSIONALI

Relazione finale in
Metodi Numerici per la Grafica

Relatore:
Prof. Damiana Lazzaro

Presentata da:
Federico Fucci

Sessione 2
Anno Accademico 2011 · 2012

Indice

Elenco delle figure

Introduzione

L'utilizzo di un *software* CAD all'interno di una qualsiasi azienda rende possibile una accelerazione nei tempi di prototipazione. Inoltre, grazie ad una prototipazione virtuale, è possibile ridurre, se non evitare completamente, la costruzione di prototipi fisici. Tutto ciò si traduce in ovvi vantaggi sia in termini di tempo che di costo, tanto nella fase di progettazione quanto in quella di assemblaggio. La presente tesi si inserisce in questo contesto ed ha per obiettivo quello di progettare e realizzare un *software* CAD, utilizzabile da parte del cliente, che permetta l'assemblaggio personalizzato di componenti tridimensionali, allo scopo di poter fare l'ordine dei componenti che gli servono per realizzare il suo progetto, della dimensione appropriata, limitando così i tempi di attesa. Il *software* è stato realizzato in Dev C++, facendo uso delle Windows API per la gestione delle finestre e delle librerie grafiche OpenGL. L'applicativo realizzato permette di importare componenti tridimensionali, per i quali sono definiti i vertici sui quali devono essere imposti i vincoli di assemblaggio e di scegliere il tipo del vincolo tra *Against* (cioè che facce piane di oggetti diversi si tocchino) , *Fits* (che consiste nellimporre che un perno cilindrico si incastri allinterno di una cavità) e *Coplanar* (che consiste nel richiedere che due facce piane giacciano sullo stesso piano). Per definire matematicamente i vincoli di assemblaggio si sono utilizzati i risultati del metodo proposto in [1], che a partire dai vincoli d'assemblaggio ben condizionati tra un componente base ed uno di assemblaggio costruisce direttamente in una matrice 4x4 che determina l'orientamento e la posizione del pezzo che si dovrà incastrare in relazione al sostegno. Seguendo questa modalità si ricava la matrice 4x4 calcolando direttamente una matrice di rotazione T_R e una matrice di traslazione T_L, le quali definiscono rispettivamente l'orientamento e la collocazione del pezzo che verrà assemblato. La tesi è così organizzata: Nel Capitolo1 saranno descritte

le motivazioni che hanno portato a progettare una sorta di CAD, ovvero i pregi di queste applicazioni e quali vantaggi possano portare ad una azienda, citando diversi campi di applicazione. Nel Capitolo 2 saranno trattati i vincoli matematici alla base degli assemblaggi presentati in [1]. Dopo averli analizzati in dettaglio richiameremo brevemente le trasformazioni geometriche affini, sia nel caso planare che nello spazio a tre dimensioni. Il Capitolo 3 descrive brevemente tutte le tecnologie utilizzate, direttamente ed indirettamente, per lo sviluppo del progetto: IDE, librerie e il *software* grafico utilizzato per la modellazione. Il Capitolo 4 è riservato alla descrizione dell'applicativo, trattando sia i dettagli tecnici legati alla sua implementazione che le varie funzionalità messe a disposizione dell'utente. Infine, nel Capitolo 5 sarà descritto l'applicativo in esecuzione, dimostrando con delle esemplificazioni e degli *screenshot* le varie funzionalità descritte nel capitolo precedente.

Capitolo 1

Il CAD a supporto dell'assemblaggio

In questo capitolo valuteremo i vantaggi che un *software* CAD può apportare alla progettazione e produzione di un prodotto industriale. Faremo anche una breve panoramica di alcuni campi di applicazione per cui il progetto sviluppato potrebbe essere utilizzato.

1.1 CAD a supporto della progettazione[10]

Con il termine CAD, *Computer-Aided Design* ovvero Progettazione Assistita dal Calcolatore, si indica la branca dell'informatica volta all'utilizzo di tecnologie *software*, in particolar modo della *computer graphics*, a supporto di tecniche progettuali di prodotti. L'obiettivo che si raggiunge per mezzo di strumenti di tipo CAD è la creazione di modelli del manufatto in esame, del quale si fornisce, nella maggioranza dei casi, una rappresentazione 3D.

Esistono due tipologie di sistemi CAD:

- CAD *Orizzontali*
  Sono sistemi che si prefiggono di poter essere utilizzati in un gran numero di domini. Possiederanno primitive molto generiche che potranno essere utilizzate in più ambiti differenti.

- CAD *Verticali*
  Sono sistemi che operano in un dominio ristretto; dato un particolare contesto

applicativo si focalizzano su di esso, fornendo primitive e strumenti *ad hoc* per risolvere problematiche specifiche.

Entrambe le tipologie di sistemi CAD hanno pregi e difetti; il progetto da noi sviluppato ricade nella seconda categoria, in quanto si è immaginato di soddisfare alle richieste di un mobilificio. Come si potrà vedere nel capitolo dedicato alla presentazione dell'applicativo in esecuzione, le funzionalità a disposizione dell'utente saranno fortemente legate alla possibilità di importare e modificare modelli esistenti.

1.1.1 Virtualizzazione della prototipazione

Un *software* CAD può rivelarsi uno strumento di importanza focale per una qualsiasi azienda; per mezzo di tale strumento, infatti, si rende possibile una accelerazione nei tempi di prototipazione. Inoltre, grazie ad una prototipazione virtuale, è possibile ridurre, se non evitare completamente, la costruzione di prototipi fisici. Tutto ciò si traduce in ovvi vantaggi sia in termini di tempo che di costo, tanto nella fase di progettazione quanto in quella di assemblaggio.

1.1.2 CAD a supporto della produzione

Ogni azienda deve, per aumentare la propria competitività sul mercato, fare dello sviluppo di nuovi prodotti e offerte il proprio obiettivo primario. Gli approcci risolutivi per questa problematica sono legati strettamente a quella della gestione dei materiali necessari per la produzione dei beni di consumo. Due sono le logiche, una di tipo *push* ed una di tipo *pull*. Mentre la prima, a partire da previsioni derivate dalla distinta di base, stabilisce la quantità di materiale necessario alla produzione, la seconda lascia al cliente il compito di richiedere il materiale necessario a soddisfare le proprie richieste.

Sfruttando una logica *pull*, quindi, l'obiettivo dell'impresa si modifica nella generazione di ″valore per il cliente″. Sempre mantenendo l'attenzione focalizzata su colui che sarà il fruitore del bene prodotto, bisogna ricordare che il cliente finale non è disposto a pagare più del prezzo che riconosce al prodotto.

Qui si colloca l'importanza di un CAD come quello proposto; permettendo una prototipazione tridimensionale, veloce ed economica risolve tutte le problematiche citate in

precedenza. Infatti applicando modifiche all'oggetto virtuale si riescono ad evitare costi aggiuntivi, che sarebbero inevitabili a fronte di quegli ″sprechi″ causati da movimentazioni delle scorte dei materiali, dagli scarti e dalle attese, e che risultano ingiustificati agli occhi del cliente, perché non generano un valore percepito da quest'ultimo. Inoltre è possibile registrare dati di modifiche apportate ad ogni singolo oggetto per poter stimare con esattezza quanto materiale in più o in meno risulti necessario a partire dalle quantità richieste per un bene standard.

1.2 Un mondo di oggetti complessi

Nella vita di tutti i giorni ognuno di noi è circondato da decine, se non centinaia, di oggetti. Questi ci semplificano la vita, permettendoci di compiere col minimo sforzo una gran mole di lavoro; molte volte ci esonerano da tutte le fatiche, prendendo completamente il nostro posto nello svolgimento delle mansioni quotidiane, o semplicemente ci accompagnano dal momento in cui apriamo gli occhi al mattino, a quello in cui li richiudiamo la sera.

Siano essi semplici o complessi, sono quasi tutti accomunati dalla stessa caratteristica: ognuno di essi è il risultato di ore e ore di progettazione; in particolare, tanto più l'oggetto risulta complesso, tanto è maggiore la probabilità che esso sia il risultato della composizione di più parti semplici. Ecco che diventa interessante la possibilità di poter gestire in maniera semiautomatica la composizione di tali oggetti a partire da primitive di base.

1.3 Arredamento

L'idea di ammobiliare in maniera interattiva una stanza, o un'intera abitazione, in un ambiente virtuale è già ampiamente utilizzata da molti grandi aziende del settore. Con il metodo che verrà presentato, oltre ad arredare secondo i desideri del cliente i vari locali, sarà possibile creare in maniera interattiva dei mobili personalizzati, scegliendo tra più componenti disponibili; questo è un ulteriore vantaggio oltre a quello banale della

relativa semplicità con cui è possibile ″incastrare″ tra loro i vari oggetti sulla base di punti d'assemblaggio che li caratterizzano.

1.4 Progettazione

Come accennato in precedenza, una possibile applicazione consiste nella produzione di un *software* di *design* che consenta di accelerare i tempi di progettazione di oggetti complessi, lasciando da modellare solo componenti più semplici.

Una applicazione di questo tipo è sicuramente la più versatile e col maggior numero di campi d'utilizzo, in quanto permette di soddisfare le esigenze che accomunano anche settori molto diversi: semplicità, funzionalità e velocità d'impiego. Non necessariamente occorre pensare all'applicazione come allo strumento di progettazione finale, potrebbe essere utilizzata come un veloce supporto all'ideazione del prodotto: visualizzando simulazioni in un ambiente virtuale tridimensionale volumetrie e ingombri è possibile effettuare studi propedeutici alla produzione in maniera efficace, semplice e, soprattutto, economica.

1.5 Industria del Videogioco

Negli ultimi anni la concezione del videogioco è profondamente mutata: l'enorme sviluppo parallelo di *hardware* dedicati e *software* di grafica ha trasformato l'esperienza videoludica in quello che può essere definita la visione di un film interattivo. Molto spesso i personaggi di questi *software* sono eroi, antichi o moderni, in possesso di equipaggiamenti che, all'atto pratico, non fanno altro che modificarne il modello. Un'ulteriore possibile applicazione potrebbe proprio essere quella di *software* per la creazione di nuovi personaggi, per velocizzare la creazione dei modelli che saranno usati dal videogioco; nei *videogame* degli ultimi anni non era insolito ritrovare sezioni dedicati alla personalizzazione: il giocatore poteva creare livelli di gioco secondo il proprio gusto estetico, così come aggiungere nuovi personaggi. Distribuendo un *software* basato sulle idee di assemblaggio presentate, sarebbe possibile soddisfare in maniera efficiente e semplice tutta la parte descritta.

Capitolo 2

Tipi di assemblaggio e vincoli matematici

In questo capitolo valuteremo la teoria alla base delle tecniche implementate. Esporremo in dettaglio i vincoli proposti dai ricercatori J. Kim, K. Kim, K. Choi e J. Y. Lee[1], ovvero le tre condizioni *Against*, *Fits* e *Coplanar*. A fine capitolo riepilogheremo brevemente le trasformazioni geometriche sia nel caso planare che nel caso spaziale. Nei vari esempi ci riferiremo sempre a due oggetti semplici che verranno assemblati per formarne uno complesso; in particolare denoteremo come componente *mate* il componente che verrà assemblato sull'altro, indicato con il termine componente *base*.

2.1 Motivazioni del metodo

Operare in uno spazio tridimensionale implica l'utilizzo di molta matematica. L'assemblaggio che stiamo ricercando, infatti, non è altro che una trasformazione geometrica che, se applicata ad un oggetto designato, lo riposiziona in maniera corretta rispetto a tutti gli altri; questa, ricordiamo, può essere espressa da una matrice quadrata di trasformazione 4x4. Moltissimi sono i metodi proposti, la prima domanda che sorge è secondo quale criterio favorire un metodo rispetto ad un altro.

Moltissime metodologie, decisamente eleganti, sono state proposte per modellare il problema dell'assemblaggio in altrettanti articoli di ricerca. Diversi approcci si basano

su procedure numeriche, in cui i vincoli di assemblaggio sono tipicamente rappresentati da un insieme di equazioni non lineari, le quali sono risolte utilizzando una procedura iterativa di Newton-Raphson modificata [2, 3, 4].

Questo approccio comporta, per la sua strutturazione, notevoli svantaggi: non garantisce soluzione ogni volta e, inoltre, le soluzioni finali sono spesso dipendenti dai valori iniziali.

Altri metodi si basano su procedure algebriche, in cui ogni operazione di assemblaggio viene suddivisa in una sequenza di operazioni di rotazione e traslazione, utilizzando la riduzione dei gradi di libertà (DOF). Quindi, la matrice di trasformazione per determinare l'orientamento relativo fra il componente e il supporto è ottenuta come prodotto di matrici di traslazione e rotazione. In un approccio del genere l'insieme delle operazioni di traslazione e rotazione è ottenuto tipicamente diminuendo in maniera sequenziale i DOF per mezzo di una procedura di consultazione di una tabella. I DOF correnti e il vincolo d'assemblaggio successivo sono gli *input*, mentre i DOF risultanti e le operazioni di rotazione e traslazione che soddisfano i vincoli stessi gli *output* [5, 6, 7, 8, 9]. Queste tabelle sono, tuttavia, spesso molto grandi.

Alla luce degli svantaggi illustrati si è scelta una strada diversa per approcciarsi al problema.

2.2 Metodo Usato

Il metodo studiato in dettaglio e applicato [1] prende come *input* vincoli d'assemblaggio ben condizionati tra un componente base ed uno di assemblaggio e li trasforma direttamente in una matrice 4x4 che determina l'orientamento e la posizione del pezzo che si dovrà incastrare in relazione al sostegno. Seguendo questa modalità si ricava la matrice 4x4 calcolando direttamente una matrice di rotazione T_R e una matrice di traslazione T_L, le quali definiscono rispettivamente l'orientamento e la collocazione del pezzo che verrà assemblato.

2.3 Vincoli d'assemblaggio dei vari metodi

I metodi esistenti prevedono tre tipologie principali di vincoli d'assemblaggio:

- Distanza: si considera la distanza fra i componenti che si dovranno assemblare.
- Angolo: si specifica l'angolo tra i componenti assemblati.
- Allineamento: si specifica se i componenti siano allineati sullo stesso lato o dal lato opposto del piano.

Le condizioni considerate nel metodo [1] sono di Allineamento, in particolare le indicheremo con i nomi *Against*, *Fits* e *Coplanar*. Un insieme di vincoli d'assemblaggio ben condizionati definisce in maniera univoca la matrice di trasformazione 4x4 che determina orientamento e collocazione relativi al componente *mate*. Descriveremo ora nel dettaglio le tre tipologie di vincolo; con gli apici b, m, mr e ma indichiamo rispettivamente il componente di base, il componente che dovrà essere assemblato, lo stesso componente dopo la rotazione e dopo l'incastro.

2.4 Against

La condizione di *Against* si ha tra due facce piane e richiede che le stesse si tocchino vicendevolmente (Fig. 2.1). Le facce designate, ombreggiate in figura, sono le facce che devono essere accostate. Ogni faccia è definita dal suo vettore normale N e da uno qualsiasi dei punti che giacciono su essa, espresso in termini di coordinate locali.

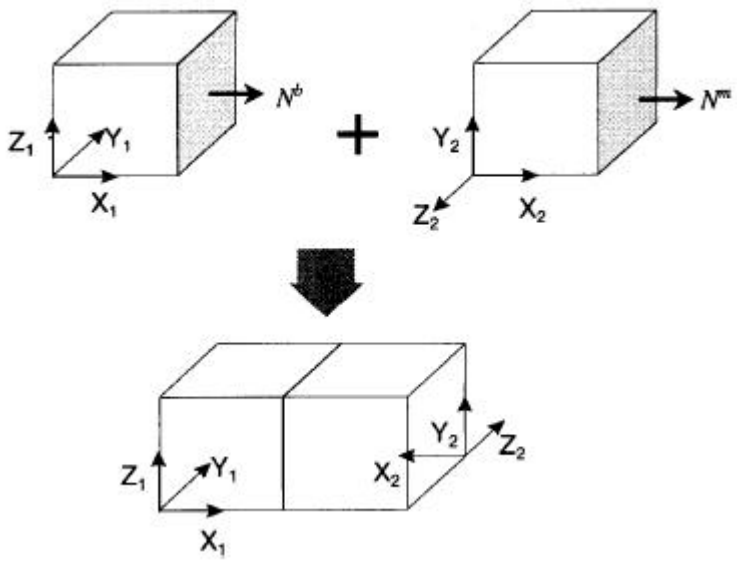

Figura 2.1: Condizione *Against*

Questa condizione si ottiene esprimendo i vincoli per mezzo delle espressioni seguenti:

$$N^b = \begin{bmatrix} N_x^b \\ N_y^b \\ N_z^b \end{bmatrix} = - \begin{bmatrix} N_x^{ma} \\ N_y^{ma} \\ N_z^{ma} \end{bmatrix} = -N^m$$

$$\begin{bmatrix} N_x^b & N_y^b & N_z^b \end{bmatrix} \begin{bmatrix} P_x^b - P_x^{ma} \\ P_y^b - P_y^{ma} \\ P_z^b - P_z^{ma} \end{bmatrix} = 0$$

Queste espressioni richiedono che le due facce selezionate siano opposte l'una all'altra, e che siano portate a giacere sullo stesso piano.

2.5 Fits

La condizione *Fits* si ha fra due facce cilindriche; una faccia di un perno e una faccia di una cavità, come mostrato in figura (Fig. 2.2). La condizione di *Fits* è soddisfatta richiedendo che l'asse centrale del perno e della cavità siano paralleli e un punto P^m sull'asse del componente che dovrà essere assemblato giaccia sull'asse del componente *base*. Un asse è definito da un vettore direzione unitario e da un punto su di esso. L'asse della cavità è specificato da un punto P^b e da un vettore direzione unitario N^b definito in termini del suo sistema di coordinate locali. Similmente, l'asse del perno è specificato da un punto P^m e da un vettore direzione unitario N^m in termini del suo sistema di coordinate locali. Perciò le equazioni dei vincoli di questa condizione sono esprimibili come segue:

$$N^b = \begin{cases} N^{ma}, & \text{per condizioni } \textit{Fits} \text{ allineate} \\ -N^{ma}, & \text{per condizioni } \textit{Fits} \text{ antiallineate} \end{cases}$$

$$\frac{P_x^{ma} - P_x^b}{N_x^b} = \frac{P_y^{ma} - P_y^b}{N_y^b} = \frac{P_z^{ma} - P_z^b}{N_z^b}$$

Notiamo come siano fornite, da quest'ultima condizione, tre combinazioni di equazioni per ogni condizione *Fits*; in generale una di queste equazioni è ridondante dal momento che la relazione fornisce solo due equazioni indipendenti, piuttosto che tre. Comunque è necessario considerarle tutte e tre per coprire quei casi in cui la linea centrale di una componente viene ad essere parallela a una degli assi coordinati cosicchè la relazione restituisce solo due equazioni.

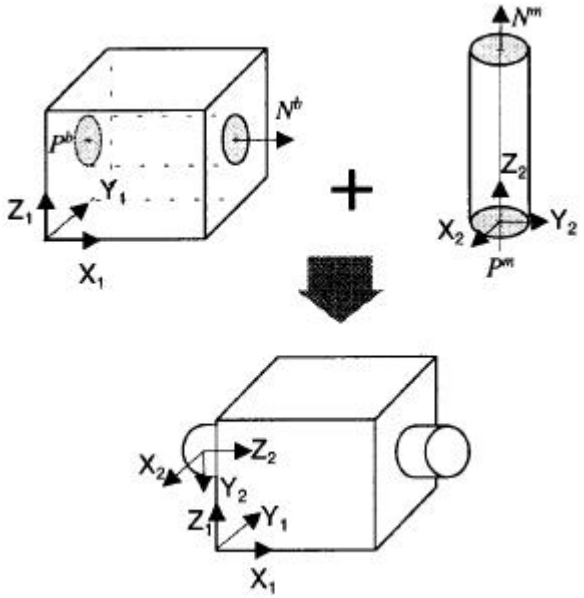

Figura 2.2: Condizione *Fits*

2.6 Coplanar

La condizione *Coplanar* si ha tra due facce piane quando esse giacciono sullo stesso piano, come in figura (Fig. 2.3). Condizione simile al vincolo *Against* eccettuato il fatto che si richiede che i due vettori normali N^b e N^m puntino nella stessa direzione. Quindi tali vincoli possono essere espressi come

$$N^b = \begin{bmatrix} N_x^b \\ N_y^b \\ N_z^b \end{bmatrix} = \begin{bmatrix} N_x^{ma} \\ N_y^{ma} \\ N_z^{ma} \end{bmatrix} = N^m$$

$$\begin{bmatrix} N_x^b & N_y^b & N_z^b \end{bmatrix} \begin{bmatrix} P_x^b - P_x^{ma} \\ P_y^b - P_y^{ma} \\ P_z^b - P_z^{ma} \end{bmatrix} = 0$$

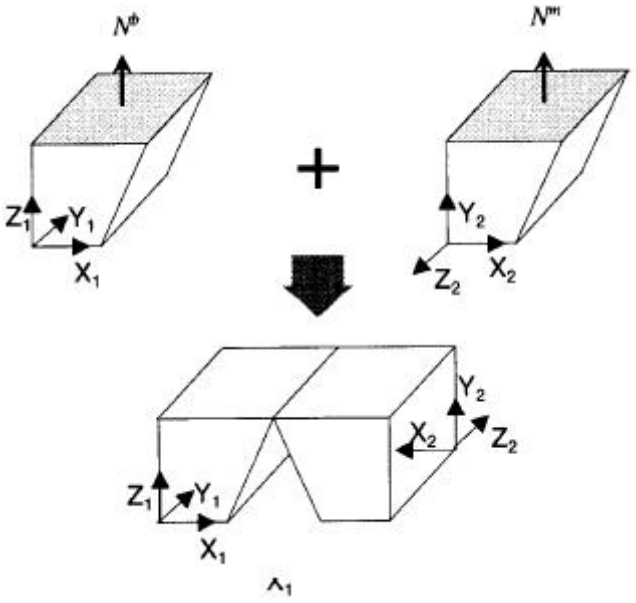

Figura 2.3: Condizione *Coplanar*

2.7 Risoluzione dei vincoli d'assemblaggio 3D

L'orientamento e la posizione relativi dei componenti da assemblare in relazione al sostegno sono rappresentati da una matrice di trasformazione 4x4. La matrice di trasformazione può essere scritta come

$$T = \begin{bmatrix} R_{1x} & R_{2x} & R_{3x} & L_x \\ R_{1y} & R_{2y} & R_{3y} & L_y \\ R_{1z} & R_{2z} & R_{3z} & L_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & R & & L \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Risulta evidente che la matrice di trasformazione T possa essere determinata calcolando in maniera indipendente due sottomatrici: una sottomatrice di rotazione R 3x3 e una sottomatrice di traslazione L 3x1. La matrice di trasformazione T potrà poi essere espressa come prodotto di una matrice di traslazione T_L e di una matrice di traslazione T_R come mostrato in seguito:

$$T = T_L T_R = \begin{bmatrix} 1 & 0 & 0 & L_x \\ 0 & 1 & 0 & L_y \\ 0 & 0 & 1 & L_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{1x} & R_{2x} & R_{3x} & 0 \\ R_{1y} & R_{2y} & R_{3y} & 0 \\ R_{1z} & R_{2z} & R_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Nella procedura di risoluzione studiata le due matrici sono determinate sequenzialmente. Si determina in un primo momento T_R dalle relazioni rotazionali tra le parti che dovranno essere assemblate; dopodichè si ottiene T_L dalle relazioni traslazionali tra il componente *base* e quello *mate*, dopo aver riposizionato quest'ultimo applicandogli la matrice di rotazione T_R.

2.7.1 Calcolo della sottomatrice rotazionale R

Come già affermato, le condizioni *Against*, *Fits* e *Coplanar* sono usate per vincolare le rotazioni dei modelli nell'assemblaggio. Ognuna di queste condizioni di assemblaggio è associata ad una coppia di vettori direzione, che possono essere allineati (quando l'angolo che essi formano è di 0) o antiallineati (quando l'angolo fra essi è 180). I vettori direzione associati sono i vettori normali unitari delle facce dei componenti *base* e *mate* per i vincoli *Against* e *Coplanar*, e il vettore direzione dell'asse dei componenti cavità e perno che vengono considerati dalla condizione *Fits*. Date due coppie indipendenti di vettori direzione, $(N_1^b, N_1^m) e (N_2^b, N_2^m)$, dai vincoli ben condizionati di assemblaggio tra i due componenti in esame, le equazioni associate con la rotazione dei componenti sono espresse come

$$N_1^{mr} = RN_1^m$$

$$N_2^{mr} = RN_2^m$$

$$where N_i^{mr} = \begin{cases} N_i^b, & \text{per condizioni allineate} \\ -N_i^b, & \text{per condizioni antiallineate} \end{cases}$$

N_i^{mr} è un vettore direzione per l'assemblaggio dopo che il componente che verrà assemblato è stato riposizionato per mezzo dell'applicazione della matrice di rotazione T_R. Quando N_3^m e N_3^{mr} sono definite come

$$N_3^m = N_1^m \times N_2^m$$

$$N_3^{mr} = N_1^{mr} \times N_2^{mr}$$

allora la relazione tra N_1^m e $N_1 mr$ diventa

$$N_3^{mr} = RN_3^m$$

È possibile riscrivere queste equazioni come

$$\begin{bmatrix} N_1^{mr} & N_2^{mr} & N_3^{mr} \end{bmatrix} = R \begin{bmatrix} N_1^m & N_2^m & N_3^m \end{bmatrix}$$

Perciò la sottomatrice rotazionale R è ottenuta

$$R = \begin{bmatrix} N_1^{mr} & N_2^{mr} & N_3^{mr} \end{bmatrix} \begin{bmatrix} N_1^m & N_2^m & N_3^m \end{bmatrix}^{-1}$$

2.7.2 Calcolo della sottomatrice traslazionale L

La sottomatrice di traslazione L è calcolata algebricamente risolvendo i vincoli d'assemblaggio associati alla traslazione dopo il riposizionamento dell'oggetto *mate* applicandogli la matrice di rotazione R. Dopo il riposizionamento, i vettori direzione del pezzo *mate* sono paralleli ai corrispondenti vettori direzione del componente *base*; un punto sul pezzo *mate* dopo l'assemblaggio è espresso come

$$P^{ma} = P^{mr} + L$$

dove P^{mr} è un punto sul componente che verrà assemblato dopo il riposizionamento di quest'ultimo, e verrà calcolato, alla luce di quanto affermato in precedenza, come

$$P^{mr} = RP^m$$

Perciò le equazioni dei vincoli associati alla traslazione sono espressi come di seguito.

1. Condizioni *Against* e *Coplanar*

Dal momento che i vettori normali dei componenti sono paralleli dopo il riposizionamento, le condizioni *Against* e *Coplanar* richiedono che un punto sulla faccia del componente *mate* giaccia sulla faccia designato per l'assemblaggio del componente *base*. Perciò queste due tipologie di vincolo possono essere espresse come di seguito

$$\begin{bmatrix} N_x^b & N_y^b & N_z^b \end{bmatrix} \begin{bmatrix} P_x^b - (P_x^{mr} + L_x) \\ P_y^b - (P_y^{mr} + L_y) \\ P_z^b - (P_z^{mr} + L_z) \end{bmatrix} = 0$$

2. Condizione *Fits*

Dal momento che gli assi dei componenti in considerazione sono paralleli dopo il riposizionamento, il vincolo *Fits* richiede che un punto P^m sull'asse del componente *mate* giaccia sull'asse del componente *base*. I vincoli si possono quindi esprimere nella maniera seguente

$$\frac{(P_x^{mr} + L_x) - P_x^b}{N_x^b} = \frac{(P_y^{mr} + L_y) - P_y^b}{N_y^b} = \frac{(P_z^{mr} + L_z) - P_x^b}{N_z^b}$$

2.8 Trasformazioni geometriche di base [12]

Per poter manipolare punti e vettori all'interno dell'universo dell'applicazione grafica occorre utilizzare trasformazioni geometriche affini. Richiamiamo brevemente come esse operino. Esse sono trasformazioni lineari

$$f(aP + bQ) = af(P) + bf(Q)$$

che preservano

- *collinearità*: i punti di una linea giacciono ancora su di una linea dopo la trasformazione
- *rapporto fra le distanze*: il punto medio di un segmento rimane il punto medio di un segmneto anche dopo la trasformazione

  Le trasformazioni geometriche permettono di traslare, ruotare, scalare o deformare

oggetti che siano stati modellati nel loro spazio di coordinate del modello, consentendone l'istanziazione con attributi diversi nello spazio di coordinate del mondo. (*Si copre quindi il passaggio tra le coordinate locali alle coordinate del mondo*

Le trasformazioni geometriche di base sono

- Traslazione
- Scalatura
- Rotazione

2.8.1 Trasformazione di traslazione

Traslare una primitiva geometrica nel piano significa muovere ogni suo punto $P(x, y)$ di d_x unità lungo l'asse X e di d_y unità lungo quello delle Y fino a raggiungere la nuova posizione $P(x', y')$ dove:

$$x' = x + d_x$$

$$y' = y + d_y$$

in notazione matriciale

$$P = \begin{bmatrix} x \\ y \end{bmatrix} P' = \begin{bmatrix} x' \\ y' \end{bmatrix} T = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

$$P' = P + T$$

con T vettore traslazione.

2.8.2 Trasformazione di scalatura

Scelto un punto C (punto fisso) di riferimento, scalare una primitiva geometrica significa riposizionare rispetto a C tutti i suoi punti in accordo ai fattori di scala S_x lungo l'asse X e S_y lungo l'asse y scelti.

Se il punto fisso è l'origine O degli assi, la traformazione di P in P' si ottiene con

$$x' = S_x \cdot x$$

$$y' = S_y \cdot y$$

in notazione matriciale ciò si esprime

$$P' = S \cdot P$$

dove

$$P = \begin{bmatrix} x \\ y \end{bmatrix} P' = \begin{bmatrix} x' \\ y' \end{bmatrix} S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

S premoltiplica P, in quanto quest'ultimo è definito come vettore colonna

2.8.3 Trasformazione di rotazione

Fissato un punto C (pivot) di riferimento ed un verso di rotazione, ruotare una primitiva geometrica attorno a C significa muovere tutti i suoi punti nel verso assegnato in maniera che si conservi, per ognuno di essi, la distanza da C; una rotazione di θ attorno all'origine O degli assi è definita come

$$x' = x \cdot cos\theta - y \cdot sin\theta$$

$$y' = x \cdot sin\theta + y \cdot cos\theta$$

in notazione matriciale si avrà

$$P' = R \cdot P$$

dove

$$P = \begin{bmatrix} x \\ y \end{bmatrix} P' = \begin{bmatrix} x' \\ y' \end{bmatrix} R = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

2.9 Coordinate omogenee [12]

Le trasformazioni geometriche possono essere applicate in sequenza (la nostra applicazione ne è un esempio); nel caso di presenza di somme di vettori, a causa di traslazioni, e moltiplicazioni, per scalature e rotazioni, si ha una concatenazione *disomogenea* di trasformazioni.

Per risolvere tale problematica si sono introdotte le coordinate omogenee, per mezzo delle quali un punto $P(x, y)$ risulta espresso come $P(x_h, y_h, w)$ dove

$$x = \frac{x_h}{w} \quad y = \frac{y_h}{w} \quad con \quad w \neq 0$$

Due punti di coordinate (x, y, w) e (x', y', w') rappresentano lo stesso punto del piano se e solo se le coordinate dell'uno sono multiple delle rispettive coordinate dell'altro; almeno uno dei valori x, y, o w deve essere diverso da 0. Quando $w = 1$ (forma canonica) coordinate omogenee e cartesiane vengono a coincidere. Con $(x, y, w \neq 0)$ si indicano punti, con $(x, y, w = 0)$ si rappresentano vettori.

È ovvio che tutte le trasformazioni geometriche esaminate possano essere espresse per mezzo delle coordinate omogenee. Si avrà quindi:

- *Trasformazione di Traslazione*

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- *Trasformazione di Scalatura*

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- *Trasformazione di Rotazione*

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

  La rappresentazione in coordinate omogenee permette una concatenazione omogenea di trasformazioni. L'ordine di concatenazione è molto importante perchè le trasformazioni geometriche, pur essendo associative, non sono commutative. Come esempio, volendo applicare, nell'ordine, le trasformazioni T^1, T^2, T^3 e T^4 occorre calcolare la trasformazione composizione T come

$$T = T^4 \cdot T^3 \cdot T^2 \cdot T^1$$

2.10 Trasformazioni 3D [12]

Se tutte le trasformazioni nel piano possono essere rappresentate per mezzo delle coordinate omogenee da matrici 3×3, le trasformazioni nello spazio possono essere rappresentate da matrici 4×4.
Nello spazio un punto in coordinate omogenee è rappresentato da un quadrupla

$$(x, y, z, w)$$

Le trasformazioni geometriche nello spazio diventano

- *Traslazione* La matrice di traslazione nello spazio tridimensionale è una semplice estensione della matrice 2D

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- *Scalatura* La matrice di scalatura nello spazio tridimensionale è una semplice estensione della matrice 2D

$$T(d_x, d_y, d_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- *Rotazione* Più complicato è il caso della rotazione attorno ad un asse qualsiasi nello spazio tridimensionale. Ci viene in aiuto la proprietà che ogni rotazione si può ottenere come composizione di rotazioni attorno ai tre assi principali.
  - *R. attorno asse X*

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

* *R. attorno asse Y*

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

* *R. attorno asse Z*

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

Per ricondursi al caso generale è sufficiente operare nella seguente maniera:

1. Traslare l'oggetto in modo tale che l'asse di rotazione passi per l'origine O degli assi
2. Ruotare l'oggetto in modo tale che l'asse di rotazione venga a coincidere con uno degli assi principali
3. Ruotare l'oggetto nel verso e della quantità angolare richiesta
4. Applicare la rotazione inversa al passo 2
5. Applicare la traslazione inversa al passo 1

Capitolo 3

Tecnologie utilizzate

In questo capitolo effettueremo una panoramica sulle tecnologie utilizzate per lo sviluppo dell'applicativo. In particolare citeremo Dev-C++ in quanto IDE scelto, le Windows API (WINAPI) con le quali si è creata un'interfaccia a finestre, le OpenGL sfruttate per la parte grafica del progetto e Blender, *software* grafico utilizzato per la modellazione dei vari oggetti su cui si opererà.

3.1 Bloodshed Dev C++

3.1.1 L'IDE Dev C++

L'IDE utilizzato per lo sviluppo del progetto è Bloodshed Dev C++. Gratuito, distribuito sotto le condizioni della GNU *General Public License*, integra MinGW, un compilatore C e C++, ovviamente anch'esso gratuito. Sviluppato originariamente da Colin Laplace, è supportato solo da Microsoft Windows. Per i nostri fini ciò non ha costituito un problema, dal momento che si intendeva fin da subito utilizzare le WINAPI.

3.1.2 Motivazioni della scelta

Come per Blender, fattore determinante per la scelta dell'IDE di sviluppo è stata la gratuità del supporto. Il fatto di essere un codice aperto non è stato particolare irrilevante, ma anzi ha contribuito positivamente nella direzione della scelta effettuata.

3.2 Windows e OpenGL

Windows è ancora il Sistema Operativo più diffuso, sia in ambito aziendale che nel privato; la scelta è stata quindi effettuata sulla base della maggior diffusione potenziale possibile per il prodotto. Analogamente OpenGL è fortemente utilizzato ed ha, come ulteriore vantaggio, la caratteristica della gratuità.

3.3 Windows API - WINAPI

Per Windows API si intende l'insieme delle interfacce di programmazione disponibili nei sistemi operativi Windows di Microsoft.
Seguendo l'evoluzione delle varie versioni dei Sistemi Operativi, si sono avute varie evoluzioni anche in esse:

- Win16: Versione iniziale a 16bit delle Windows API
- Win32: Evoluzione a 32bit
- Win32s: sottoinsieme di Win32 da installare su versioni di Windows a 16bit per eseguire applicazioni compilate a 32bit
- Win64: Evoluazione a 64bit per le versioni di Windows destinate ai processori a 64bit

3.3.1 Struttura delle Windows API

Le API consistono in un insieme di funzioni in linguaggio C implementate in una DLL (*Dynamic-link library*). Nonostante esse siano scritte in C con del linguaggio assembly inline per motivi di prestazione, presentano un modello complessivo che si avvicina a quello proposto dal paradigma a oggetti.

La struttura base delle Windows API, rimasta pressocchè invariata nel tempo, prevede tre gruppi principali di API: *kernel*, *GDI* e *user*.

3.3.2 API Kernel

Le API *kernel* forniscono alle applicazioni un'interfaccia di alto livello ai servizi del kernel del sistema operativo.

3.3.3 API GDI

Le API *GDI*(*Graphics Device Interface*) costituiscono la libreria grafica dei sistemi di Windows. GDI virtualizza tutti i dispositivi grafici (*monitor*, stampanti, *plotter*) in modo da avere un'interfaccia omogenea (detta *Device Context*) tra le differenti tipologie di dispositivi. Inoltre GDI permette di creare e manipolare una serie di oggetti grafici, tra cui font, penne, pennelli, bitmap e così via.

3.3.4 API User

Le API *User* forniscono utili servizi di interfaccia grafica, basati sui concetti di ″finestra″ e di ″messaggio″.

3.3.5 Le WINAPI nel progetto

Nel progetto si sono utilizzate sia le API *GDI* che le API *User*; le prime sono state utilizzate per recuperare il *Device Context* affinchè si potesse poi operare con le funzioni messe a disposizione da OpenGL. Molto sfruttate sono state, invece, le funzionalità messe a disposizione dalle API *user*. Si è infatti creata un'applicazione a finestre, in particolare un'applicazione MDI (*Multiple Document Interface*), seguendo il paradigma evento-messaggio, ovvero ad ogni evento, generato dall'utente o dal sistema, si associa un messaggio che, inviato alla relativa finestra, verrà elaborato a seconda della sua natura.

3.3.6 Perchè WINAPI e non MFC?

Le MFC (*Microsoft Foundation Class Library*) sono una libreria che incapsula porzioni delle funzioni delle Windows API in classi C++, rendendo in alcune situazioni più intuitiva la progettazione in un modello *object oriented*. Si è scelto comunque di non

utilizzarle per due motivazioni principali:

- Le MFC sono molto più lente rispetto alle Windows API

  le Windows API seguono implementazioni ottimizzate per processori di vecchie generazioni, questo fa sì che, con le macchine attuali, si ottengano prestazioni veramente ottime. Questo vantaggio si tende a perdere con le MFC.

- Le MFC non supportano correttamente applicazioni MDI

  la implementazione più lineare di un'applicazione MDI con le MFC prevede che ad ogni finestra sia associata la medesima ″intelligenza″, ovvero che ogni finestra risponda alla stessa maniera ai messaggi che riceve in seguito al verificarsi di un evento. Dal momento che nell'applicazione prodotta non si voleva questo, ma si desideravano comportamenti diversi per diverse finestre, piuttosto che creare classi contorte per la realizzazione di questo obiettivo, si è deciso di non utilizzarle.

3.4 OpenGL

OpenGL è una specifica che definisce una API per più linguaggi e per più piattaforme per scrivere applicazioni che producono computer grafica 2D e 3D. È lo standard di fatto per la computer grafica 3D in ambiente Unix.

A livello più basso OpenGL è una specifica, ovvero si tratta banalmente di un documento che descrive un insieme di funzioni ed il comportamento che ognuna di esse deve produrre. Da questa specifica i produttori di *hardware* creano implementazioni, ovvero librerie di funzioni create rispettando quanto riportato dalla specifica OpenGL, facendo uso dell'accelerazione *hardware* dove possibile. Esistono implementazioni efficienti di OpenGL - che sfruttano in modo più o meno completo le GPU - per Microsoft Windows, Linux, molte piattaforme Unix, Playstation 3 e Mac OS.

3.4.1 Struttura

OpenGL assolve due compiti fondamentali:

- nascondere la complessità di interfacciamento con acceleratori 3D differenti, offrendo al programmatore una API unica ed uniforme;

- nascondere le capacità offerte dai diversi acceleratori 3D, richiedendo che tutte le implementazioni supportino completamente l'insieme di funzioni OpenGL, ricorrendo ad una emulazione *software* se necessario.

Il compito di OpenGL è quello di ricevere primitive geometriche (punti, linee, poligoni) e di convertirli in pixel, seguendo tutto il processo di rasterizzazione. Questo si ottiene attraverso l'esecuzione di una *pipeline* grafica nota come *OpenGL State Machine*. La maggior parte dei comandi OpenGL forniscono primitive alla *pipeline* grafica o istruiscono la *pipeline* stessa su come elaborarle.

La versione utilizzata è la 2.1.

3.5 Blender

Per la modellazione dei componenti è stato utilizzato Blender 2.49, *software* di modellazione grafica 3D open source.

3.5.1 Storia[Blender, Storia]

Blender è stato sviluppato come applicazione interna dallo studio di animazione olandese NeoGeo e Not A Number Technologies(NaN). Primo autore è Ton Roosendal, già sviluppatore nel 1989 del *ray tracer* Traces per Amiga.
É Roosendal a fondare nel Giugno 1998 NaN e a distribuire e sviluppare il *software*.
Il programma era inizialmente distribuito come *shareware* finchè nel 2002 la NaN fece bancarotta. Oggi Blender è un *software* gratuito ed *open source* ed è, eccezion fatta per due impiegati *part-time* e due a tempo pieno al Blender Institute, completamente sviluppato dalla comunità.

3.5.2 Motivazioni della scelta

Il fatto di essere un *software* gratuito ha reso Blender il miglior candidato ad essere il programma di modellazione per le necessità del progetto sviluppato. Non ha nulla da invidiare a *software* commerciali e permette di lavorare anche ad un livello alto. Altro fattore fondamentale che ha guidato la scelta verso di esso, è stato il fattore del codice aperto e della possibilità di aggiungere facilmente *script* per una sua estensione per mezzo del linguaggio Python. In un'ottica del genere è infatti possibile pensare allo sviluppo di uno *script* che permetta l'importazione e l'esportazione di modelli in formato Wavefront OBJ Esteso (si vedano i capitoli seguenti), con la possibilità di inserire informazioni necessarie alla corretta esecuzione del programma direttamente in fase di modellazione.

3.6 Un esempio di modellazione

Possiamo riscontrare quattro fasi principali durante la modellazione di un oggetto per le nostre finalità

- Realizzazione di uno schizzo o formazione di un'idea di partenza
- Modellazione vera e propria
- *Texturing*
- Esportazione del modello nel formato Wavefront OBJ

Uno schizzo di partenza può facilmente essere importato in Blender per essere usato come immagine di riferimento. Tale bozza risulta utile, ma non è necessaria, specie se si vogliono eseguire modellazioni semplici come quella in esempio;
Attraverso gli strumenti di estrusione e scalatura, a partire da un cubo abbiamo modellato una semplice sedia. Dal momento che la struttura è simmetrica rispetto ad un asse parallelo a quello delle y (in realtà dell'asse z in Blender, poichè diversamente da altri *software* di modellazione lavora con lo spazio ruotato di 90 rispetto all'asse delle ascisse), si è sfruttata la funzionalità *mirror*: lavorando solo su di un lato del modello tale *modifier* ricrea le operazioni anche su quello speculare, velocizzando e aumentando la precisione di modellazioni come quella eseguita.

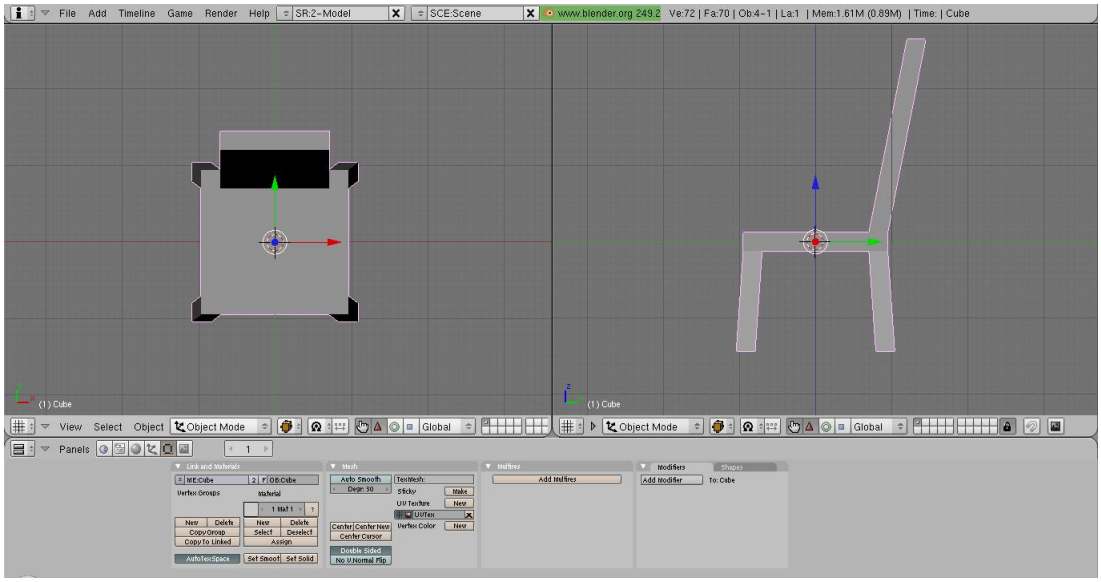

Figura 3.1: Modello di una sedia

Dopo aver modellato l'oggetto occorre effettuare una mappatura UV delle *texture*. Senza scendere nei dettagli tecnico-teorici, occorre assegnare ad ogni vertice che descrive la geometria del modello un vertice della immagine designata, in modo tale che esso possa essere correttamente ″ricoperto″ dalla *texture* scelta. Anche per questa operazione si sfrutta Blender, in quanto in possesso di un *image editor* integrato. É importante ricordarsi, dopo aver effettuato la mappatura UV, di assegnare un oggetto al modello che utilizzi la *texture* prescelta; solo in questo modo, infatti, durante l'esportazione si creerà un *file* mtl contenente informazioni relative a materiali e *texture*, che sarà necessario per una corretta visualizzazione dopo l'importazione nel CAD sviluppato.

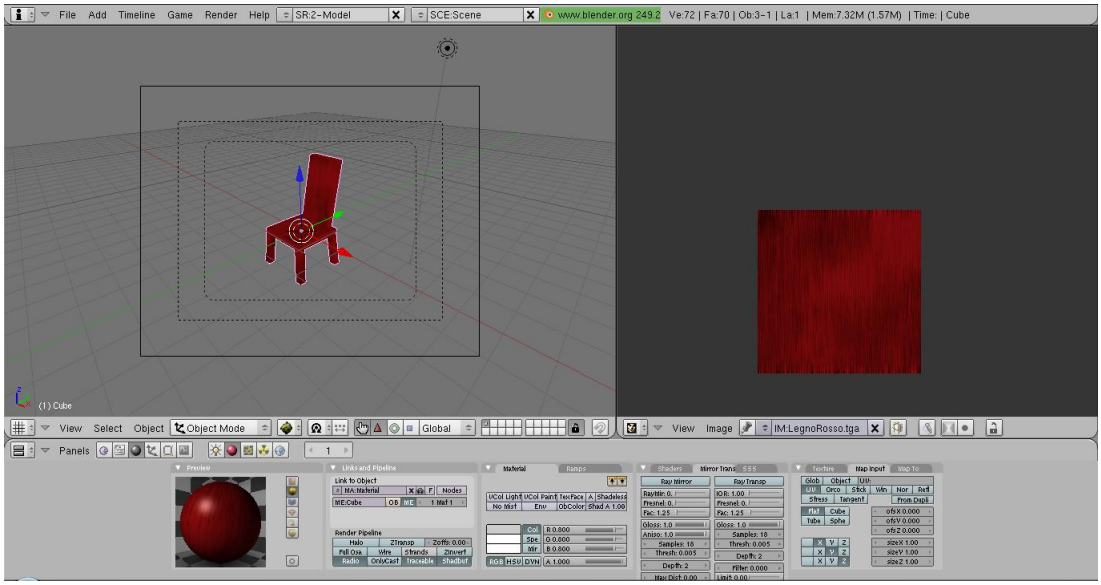

Figura 3.2: Texturing di un modello

Terzo e ultimo passaggio è quello dell'esportazione nel formato desiderato del modello creato. Dal menù *File* si selezionano la voce *Export* e in seguito la voce Wavefront(.obj) per esportare nel formato da noi voluto. L'utente deve a questo punto impostare una serie di parametri, dopo aver indicato dove desidera salvare il modello, in particolare dovrà indicare che desidera esportare le normali e le coordinate UV per il modello, rispettivamente per il corretto funzionamento del CAD - che, ricordiamo, sfrutta tali normali per la costruzione di vincoli di assemblaggio - e per una corretta visualizzazione delle *texture*.

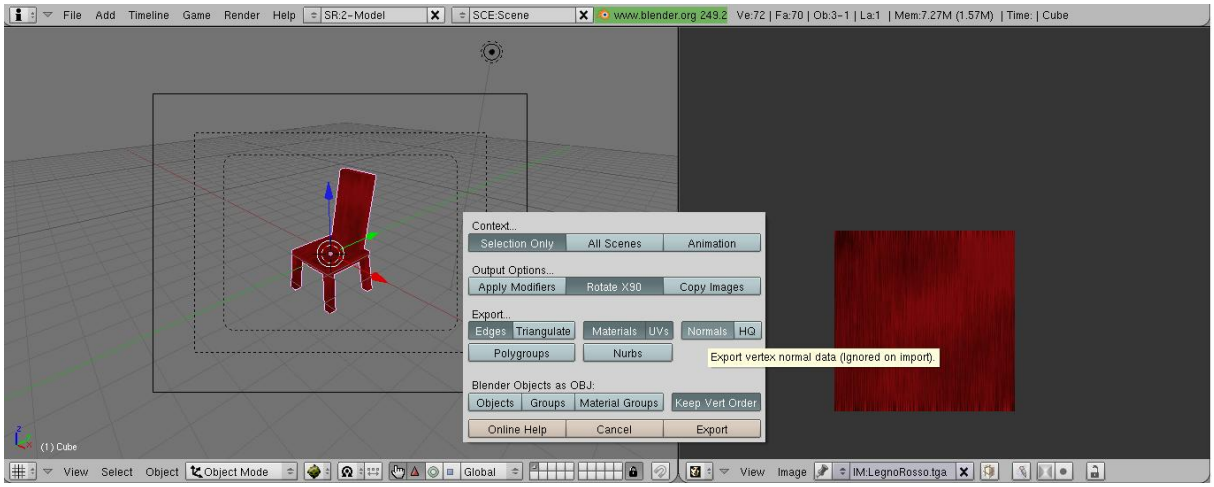

Figura 3.3: Export di un modello

3.6.1 Il modello importato

Dopo le fasi di modellazione, *texturing* ed esportazione il modello può essere importato all'interno dell'applicazione; una volta eseguita l'operazione di *import* è possibile utilizzare il nuovo oggetto per gli scopi desiderati.

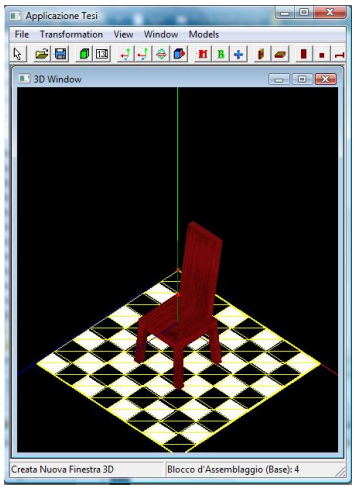

Figura 3.4: L'oggetto modellato correttamente importato

Capitolo 4

Applicativo

In questo capitolo descriveremo il programma sviluppato, soffermandoci sulle varie funzionalità messe a disposizione. Descriveremo anche qualche dettaglio tecnico inerente la progettazione, come il concetto di *Assembly Group* o come le specifiche del formato Wavefront OBJ Esteso, ottenuto per estensione del formato Wavefront OBJ standard.

4.1 Panoramica Generale

L'applicativo si presenta come un supplemento all'assemblaggio di componenti di base premodellate. Per motivi legati alla macchina su cui il codice è stato sviluppato, è stata utilizzata la versione delle WINAPI Win32.
Il progetto si è sviluppato sfruttando la peculiarità del C++ di non obbligare ad uno sviluppo ad oggetti puro; si è creata una classe per gestire i modelli che saranno poi bersaglio degli assemblaggi, mentre per svolgere altri compiti, come per esempio la gestione delle finestre, ci si è rifatti ad una programmazione di tipo imperativo. Questo riduce probabilmente l'eleganza complessiva, tuttavia ha permesso di usufruire dei pregi delle due metodologie: si è mantenuto ben separato dal resto il concetto di oggetto tridimensionale, dove in questo caso il termine ″oggetto″ è da intendersi con il significato datogli dal linguaggio naturale, mentre si sono riuscite a gestire in maniera più semplice le finestre che compongono l'applicazione stessa, senza dovere ricorrere a soluzioni che avrebbero rallentato sia la stesura del codice che l'esecuzione dell'applicativo. Ovviamen-

te, come in ogni scelta, non ci si possono aspettare solo vantaggi, ma occorre accettare anche gli inconvenienti che tale decisione comporta: nel caso particolare, solo i modelli tridimensionali potrebbero essere agevolmente esportati e riutilizzati in un progetto di diversa natura, mentre tutta la parte rimanente sarebbe in gran parte, se non del tutto, inutilizzabile in altri contesti.

La versione allo stadio attuale permette l'importazione e l'assemblaggio di più componenti, supportando il formato Wavefront OBJ e il formato Wavefront OBJ Esteso; si possono inserire punti caldi per l'assemblaggio denominati *Assembly Group* ed esportare modelli nel formato OBJ Esteso.

4.2 Struttura di base: Applicazione MDI

La struttura di base utilizzata per lo sviluppo dell'applicazione è stata quella di un applicativo MDI, ovvero *Multiple Document Interface*. Tale tipologia di applicativi, molto utilizzati in diversi ambiti, prevede di costruire una finestra principale di lavoro, la *MDI Main Frame*, la quale avrà per figlia una finestra ″contenitore″, la *MDI Child Frame*, che appunto conterrà ogni altra finestra che parteciperà all'esecuzione del programma. Sarà poi possibile inserire barre di menù e degli strumenti, sia nella *MDI Main Frame* per controllare comportamenti di tutte le sottofinestre, sia in queste ultime singolarmente, per circoscrivere l'area di effetto di una determinata operazione.

4.2.1 Tipologie delle finestre

Nell'applicazione ritroviamo cinque tipologie di finestre:

- La *MDI Main Frame*, la cui presenza è dettata dal paradigma MDI
- La *MDI Child Frame*, anch'essa necessaria per lo sviluppo di un *software* aderente al paradigma MDI
- Finestre 3D, con le quali l'utente si interfaccerà più frequentemente, durante l'esecuzione

- La finestra di *input*, per l'immissione di dati precisi per operare trasformazioni geometriche agli oggetti caricati

- La finestra *Custom*, che permette di effettuare una creazione con parametri personalizzabili di una semplice scaffalatura

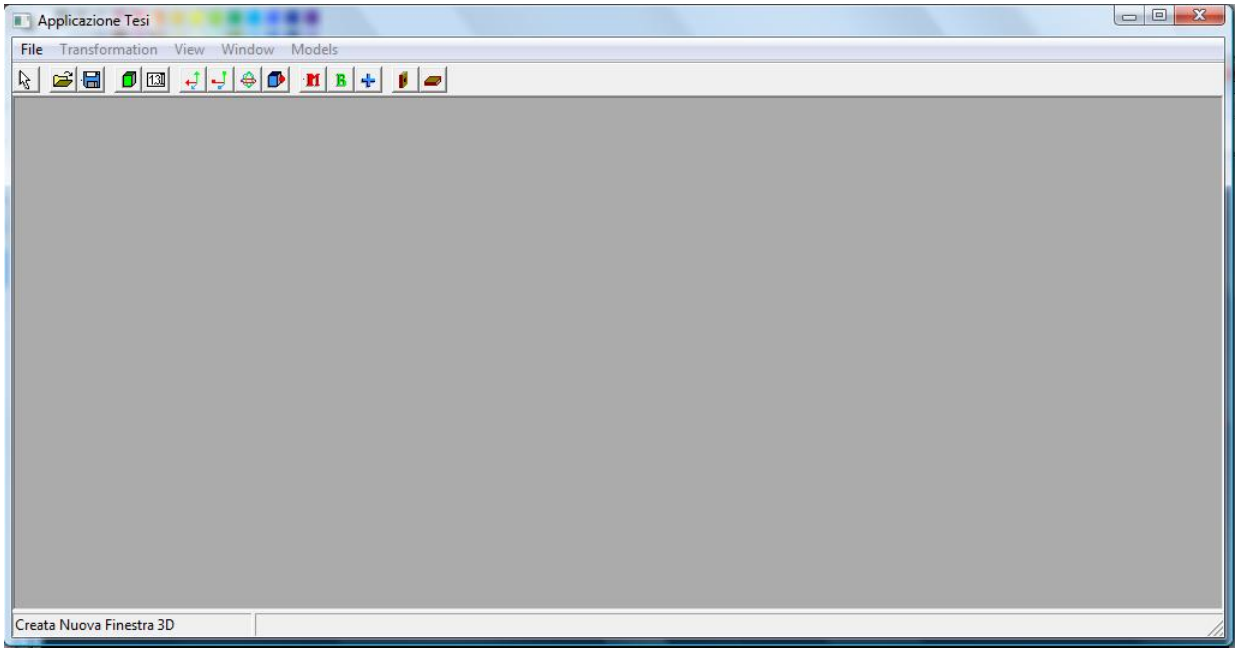

Figura 4.1: *MDI Main Frame e MDI Child Frame*

Figura 4.2: *Finestra 3D*

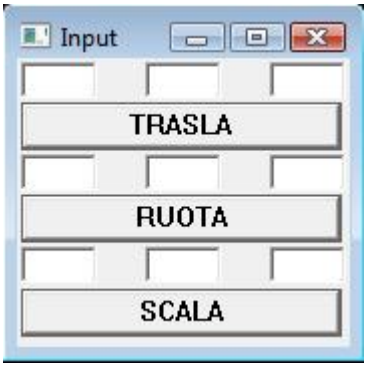

Figura 4.3: *Finestra di input*

Figura 4.4: *Finestra Custom*

4.3 Modello Tridimensionale

Ogni modello tridimensionale nel dominio della applicazione è rappresentato da un oggetto; in particolare, all'interno della classe che rappresenta il concetto di modello 3D si è incapsulata una struttura proveniente da una libreria aperta, che ha semplificato di molto il lavoro su questa parte dell'applicativo. Metodi e attributi della classe aggiungono funzionalità mancanti o implementano in maniera diversa quelle esistenti.

Conseguenza interessante, oltre che molto utile, dovuta all'incapsulamento di una *struct* che rappresenta la sola geometria del modello, è la possibilità di memorizzare un vettore di queste particolari strutture dati, in modo da memorizzare assemblaggi complessi, non perdendo le singole individualità dei componenti base.

4.4 Assembly Group

Attributo più importante a caratterizzazione dei vari modelli e il vettore degli *Assembly Group*, ovvero il vettore delle *struct* che identificano i punti caldi per l'assemblaggio. Ogni singolo *record* mantiene tre vettori, per la memorizzazione di punto e normale identificanti le facce o gli assi che si considereranno per l'assemblaggio (si veda il capitolo 2), e per l'identificazione della tipologia di vincolo. I vettori in questione hanno

profondità 2, in quanto sono necessari almeno due informazioni su vertici e normali per la costruzione della matrice di trasformazione ricercata.

```
typedef struct _AG
{
    Constraint type[NUM_CONST];
    GLint P[NUM_CONST];
    GLint N[NUM_CONST];
}AG;
```

4.5 Formato Wavefront OBJ Esteso

Il formato Wavefront o OBJ è un *file* di definizione geometrica sviluppato da Wavefront Technologies per il suo *package* di animazione *Advanced Visualizer*. Tale formato è aperto, ed è stato adottato da altri produttori di applicazioni grafiche 3D. É praticamente un formato universalmente riconosciuto. Il formato OBJ è un semplice formato-dati che rappresenta la sola geometria di un modello; essendo memorizzato in formato ASCII, è possibile prendere visione di un qualsiasi *file* OBJ anche per mezzo di un qualunque *editor* di testo.

La sua semplicità l'ha reso ideale all'utilizzo nell'applicazione creata, e, per velocizzare l'assemblaggio dei vari componenti, lo si è esteso, aggiungendo al suo interno delle parole chiave che permettessero, in fase di *import* del modello stesso, di individuare i punti cardine per i vari assemblaggi. In particolare, alla fine del *file*, si inseriscono le informazioni inerenti agli *Assembly Group* col seguente formato:

```
ag #
pC1 iP1
nC1 iN1
pC2 iP2
nC2 iN2
```

ovvero, abbiamo

- ag #: parola chiave ag (*Assembly Group*) seguita dal progressivo di tale gruppo d'assemblaggio
- pC1 iP1: indica l'indice del vertice necessario a individuare la prima condizione d'assemblaggio; C1 può essere f, a, c ad indicare, rispettivamente le condizioni di *Fits*, *Against*, *Coplanar*.
- nC1 iN1: indica l'indice della normale necessaria a individuare la prima condizione d'assemblaggio; C1 può essere f, a, c ad indicare, rispettivamente le condizioni di *Fits*, *Against*, *Coplanar*.
- pC2 iP2: indica l'indice del vertice necessario a individuare la seconda condizione d'assemblaggio; C2 può essere f, a, c ad indicare, rispettivamente le condizioni di *Fits*, *Against*, *Coplanar*.
- nC2 iN2: indica l'indice della normale necessaria a individuare la seconda condizione d'assemblaggio; C2 può essere f, a, c ad indicare, rispettivamente le condizioni di *Fits*, *Against*, *Coplanar*.

Tale estensione velocizza notevolmente il processo di lavoro, in quanto importando il modello si hanno già informazioni sui punti caldi dell'assemblaggio. Particolare interessante è che, pur aggiungendo tali informazioni ausiliarie, il modello può ancora essere letto e correttamente visualizzato da qualsiasi *software* di grafica tridimensionale che supporti l'importazione del formato obj.

4.6 Esportazione in Formato OBJ Esteso

Modificare manualmente i singoli *file* per salvare informazioni sugli *Assembly Group*, nonostante sia possibile, risulta altamente complicato e controintuitivo, oltre a portare facilmente ad errori che compromettono i futuri assemblaggi. Per questo motivo l'applicazione permette di aggiungere gruppi di assemblaggio a modelli importati e la successiva esportazione degli stessi, di modo tale che, in successive esecuzioni, si abbia sempre a disposizione il modello correttamente corredato dei vincoli necessari, pronto ad un assemblaggio rapido.

exModel
name index path matrix[16] backupMatrix[16] matrixRot[16] model vscale[3] center[4] nCenter[4] color[3] editMode numAssemblyGroups assemblyGroups numModels
readAssemblyGroups(char path[]) Model():GLMmodel* Name():GLint Center(float* newCenter = NULL):GLfloat* Matrix():GLfloat* BackupMatrix()GLfloat* exDraw(GLuint mode, BOOL vertex = FALSE) applicaRotazione(GLfloat rx, GLfloat ry, GLfloat rz, int pivot = 0) applicaRotazioneInBackUp(GLfloat rx, GLfloat ry, GLfloat rz, int pivot=0) applicaTraslazione(GLfloat dx, GLfloat dy, GLfloat dz) applicaTraslazioneInBackUp(GLfloat dx, GLfloat dy, GLfloat dz) applicaScalatura(GLfloat sx, GLfloat sy, GLfloat sz, int pivot = 0) applicaScalaturaInBackUp(GLfloat sx, GLfloat sy, GLfloat sz, int pivot = 0) confermaTrasformazione() changeEditState():BOOL

EditMode():BOOL
SetIndex(GLint newIndex)
GetIndex():GLint
NumAssemblyGroups():int
AssemblyGroups(GLint index):AG
addAssemblyGroup(int indexPt = 0);
SetNZero(GLint index);
SetNOne(GLint index);
resetTranfsormations(BOOL dontask = false):BOOL
applyTransformations()

4.7 Modalità di visualizzazione

Il programma prevede tre modalità di visualizzazione:

- *Default*: ogni oggetto è riempito di un colore standard
- *Material*: ogni oggetto è visualizzato secondo le informazioni relative al materiale che lo caratterizza
- *Wireframed Mode*: di ogni oggetto viene visualizzata solo la struttura, alleggerendo fortemente i calcoli necessari alla visualizzazione degli stessi.

4.8 Funzionamento di base

Un'esecuzione tipo prevede di caricare due o più componenti e assemblarli insieme, dopo aver eventualmente applicato qualche trasformazione agli stessi.

È possibile importare modelli utilizzando la voce *Import* del menù *Models*, oppure utilizzare l'icona rapida nella barra degli strumenti. Una finestra di dialogo permetterà la navigazione delle cartelle per selezionare l'oggetto desiderato.
Dal momento che alcuni componenti possono essere pensati come principali, sono state predisposte delle scorciatoie, per l'importazione rapida di certe componenti, di utilizzo più comune.

Una volta importati i modelli, sarà possibile selezionare ognuno di essi effettuando un *click* con il tasto sinistro del mouse. La struttura base evidenziata indica che l'oggetto è effettivamente selezionato, e si potranno a questo punto applicare ad esso le trasformazioni desiderate. Ogni oggetto potrà essere sottoposto alle trasformazioni di

- Traslazione
- Rotazione
- Scalatura

In questa fase di lavoro sarà poi possibile annullare le trasformazioni applicate ad un determinato oggetto, ricercando la voce *reset* nel menù *models*, o rendere le stesse permanenti, per mezzo del comando *apply*, sempre all'interno del menù *models*.
È di fondamentale importanza, al fine di assemblare correttamente i vari componenti, confermare ogni trasformazione che venga applicata ai componenti che prenderanno parte all'assemblaggio stesso

Il momento dell'assemblaggio è ovviamente il cuore dell'applicazione. Ogni operazione di *mating* prende parte tra due componenti singoli, ricordiamo dalla teoria matematica i due componenti *Base* e *Mate*.

Esistono due modalità per l'esecuzione di un assemblaggio

- Assemblaggio Standard

  In un assemblaggio standard occorre, come prima cosa, identificare i ruoli dei due componenti in gioco; ciò si effettua selezionando le voci *Role>Base* e *Role>Mate* dal menù *Models* o utilizzando gli appositi bottoni nella barra degli strumenti con gli oggetti selezionati. Se l'assegnamento dei ruoli è andato a buon fine, il componente *Base* risulterà colorato di verde, mentre il componente *Mate* sarà evidenziato in rosso.
  A questo punto si dovranno individuare i punti d'assemblaggio e le relative normali, operazione non necessaria se si sono importati oggetti in formato obj esteso.

Terminato questo passaggio l'assemblaggio vero e proprio si avvierà per mezzo della voce *Assembly* nel menù *Models* o, ancora una volta, per mezzo dell'apposito bottone contenuto nella *toolbar*.

- Assemblaggio Interattivo

  Un assemblaggio di tipo interattivo risultà più immediato per l'utente. Semplicemente l'utilizzatore seleziona l'oggetto che vorrà assemblare e da avvio all'operazione per mezzo dell'apposito comando nel menù *Models* o nella *toolbar*.
  Avviata l'operazione, l'oggetto selezionato diventerà rosso, ad indicare il corretto avvio della procedura. Si traslerà l'oggetto per mezzo del mouse tenendo premuto il tasto sinistro; al rilascio di questo si avrà l'assemblaggio fra esso e il più vicino oggetto nella scena.

4.9 Menù Models

Nella descrizione di un assemblaggio di base si è utilizzato numerose volte il menù *Models*; questo è dovuto al fatto che in questo menù sono contenute le operazioni che agiscono sui modelli importati, e quindi rivestono, all'interno dell'applicazione, un ruolo fondamentale.

Elenchiamo nel seguito i vari comandi di tale menù :

- Import
  Con questo comando si apre la finestra di dialogo per l'importazione di un componente; è supportato il formato Wavefront.

- Export
  Con questo comando si apre la finestra di dialogo per l'esportazione di un componente in formato OBJ esteso; è necessario avere un modello selezionato e non essere in modalità *Vertex Edit*.

- Delete
  Con questo comando si elimina permanentemente un oggetto selezionato.

- Wireframe
  Questo comando permette il passaggio da una visualizzazione *shaded* ad una visualizzazione in *wireframe*.

- *Vertex Edit*
  Con questo comando si visualizzano i vertici del modello selezionato; sarà possibile selezionarli singolarmente per marcare vari punti di assemblaggio. In questa modalità sono visibili tutti i ″gruppi d'assemblaggio″ presenti sul modello, se esso non avrà assegnato un ruolo fra *Mate* e *Base*, o quello che si sta utilizzando correntemente, in caso contrario.
  Nota: per scegliere il ″gruppo d'assemblaggio″ per un modello fornito di ruolo all'interno di un assemblaggio si possono usare i tasti '<' e '>'

- Deselect All
  Con questo comando si deseleziona ogni modello che sia stato precedentemente selezionato.

- Vertices
  Questa voce di menù apre varie opzioni per l'assegnamento delle condizioni di assemblaggio ai vari vertici di un modello selezionato.

- Role
  Questa voce di menù permette di gestire il ruolo di un componente in una operazione di assemblaggio

- Assembly
  Questo comando avvia l'assemblaggio fra i componenti *Base* e *Mate* selezionati.

- Interactive Assembly
  Questo comando avvia l'assemblaggio interattivo dei componenti. Per poter funzionare correttamente un oggetto deve essere precedentemente selezionato, e tutte le trasformazioni che può aver subito devono essere confermate.

- Apply

  Tutte le trasformazioni geometriche applicate ad un modello vengono confermate e rese permanenti. Annullando le operazioni future si arriverà all'ultima configurazione confermata in questa maniera.

- Reset

  Si annullano tutte le trasformazioni applicate ad un modello, per riportarlo all'ultima situazione confermata.

  Nota: un assemblaggio è costituito da una sequenza di trasformazioni geometriche di base; sarà perciò possibile annullare un assemblaggio con questo comando. Analogamente, per confermarlo, bisognerà usare il comando Apply.

- Custom Shelf

  Apre una finestra di dialogo, con la quale è possibile introdurre parametri per la costruzione di una libreria composta da un numero di scaffali e di mensole per ogni scaffale a piacere.

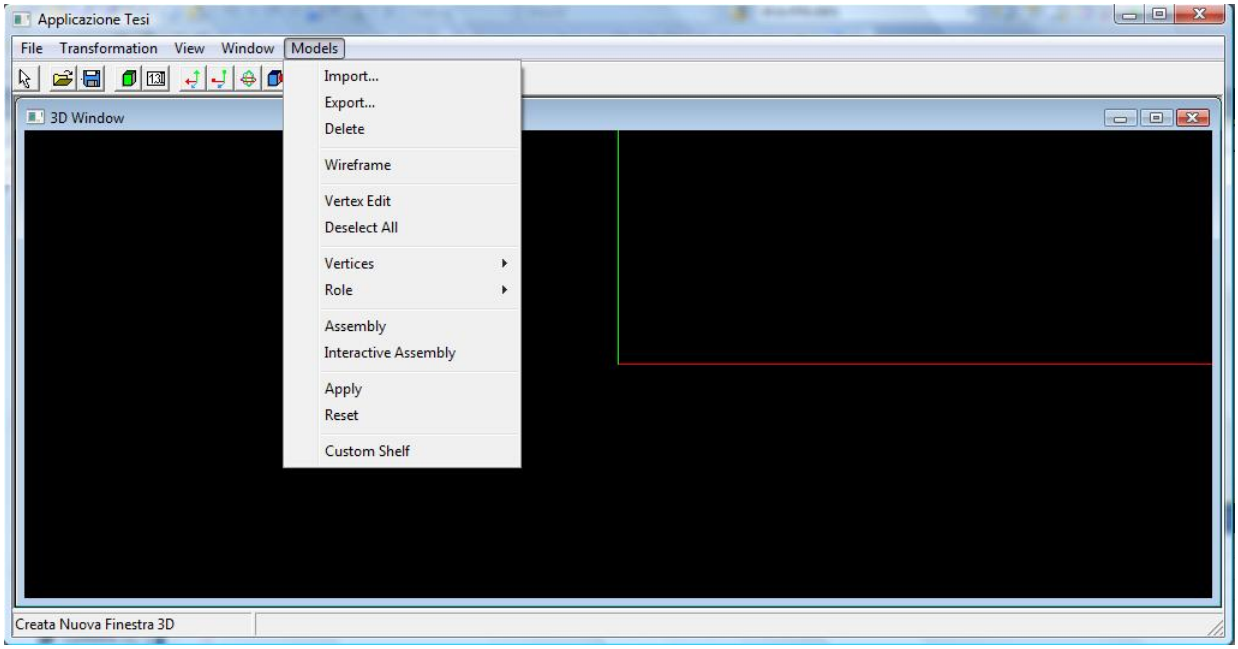

Figura 4.5: Menù Models

4.10 Toolbar e comandi rapidi

Programmi commerciali di grafica 3D sfruttano moltissimo, forse più di altre tipologie di *software*, il meccanismo dello *shortcut*; non è raro utilizzare applicativi con qualche decina di *hotkey* che risparmiano all'utilizzatore tantissimo tempo utile, che potrà quindi spendere in maniera più produttiva.
Da questo presupposto, volendo simulare un ambiente che potrebbe, previo miglioramento generale, essere utilizzato in ambito lavorativo, si sono inseriti alcuni comandi rapidi, per l'esecuzione delle operazioni principali o più ricorrenti.

Descriviamo inizialmente i bottoni della barra degli strumenti, e le operazioni ad essi collegati.

Figura 4.6: Toolbar

- Selezione
  Questo bottone riporta l'operazione corrente a selezione. Il cursore è quello di *default*. In questa modalità è possibile selezionare i vari oggetti per applicare loro le varie operazioni desiderate.

- Apri
  Questo bottone equivale alla voce *Import* nel menù *Models*. Si apre una finestra di dialogo per l'importazione di un modello tridimensionale in formato Wavefront.

- Salva
  Questo bottone equivale alla voce *Export* nel menù *Models*. Si apre una finestra di dialogo per l'esportazione del modello tridimensionale selezionato in formato Wavefront Esteso.

- Finestra 3D
  Con questo bottone si apre un'ulteriore finestra 3D di lavoro. È molto utile mantenere più finestre aperte per valutare la scena da più angolazioni.

- Finestra Input
  Apre la finestra di *input* per l'inserimento di dati precisi per l'applicazione della trasformazioni.

- Traslazione
  Avvia l'operazione traslazione. Il cursore diventa una freccia a croce. Sarà possibile traslare il modello selezionato.

- Scalatura
  Avvia l'operazione di scalatura. Il cursore diventa una lente d'ingrandimento. Sarà possibile scalare il modello selezionato.

- Rotazione
  Avvia l'operazione di rotazione. Il cursore diventa una freccia curva. Sarà possibile ruotare il modello selezionato.

- Assemblaggio Automatico
  Avvia l'operazione di assemblaggio automatico. Il modello selezionato potrà essere spostato nello spazio e, al rilascio del tasto sinistro del mouse, esso verrà assemblato automaticamente al modello più vicino.

- Designa *Mate*
  Selezionando un modello lo si designa come componente *Mate* nel successivo assemblaggio.

- Designa *Base*
  Selezionando un modello lo si designa come componente *Base* nel successivo assemblaggio.

- Assembla
  Avvia un assemblaggio standard, se sono stati indicati un modello di tipo *Base* ed uno di tipo *Mate*.

- Importa Sostegno
  Importa automaticamente un modello rappresentante un sostegno per la costruzione di una scaffalatura.

- Importa Scaffale
  Importa automaticamente un modello rappresentante uno scaffale.
- Importa Armadio
  Importa automaticamente un modello rappresentante un armadio.
- Importa Comodino
  Importa automaticamente un modello rappresentante un comodino.
- Importa Letto
  Importa automaticamente un modello rappresentante un letto.
- Importa Scrivania
  Importa automaticamente un modello rappresentante una scrivania.
- Importa Pavimento
  Importa automaticamente un modello rappresentante un pavimento.

Riportiamo ora un elenco dei principali *hotkey* utilizzati durante un'esecuzione dell'applicativo.

- W
  Passa dalla modalità *shaded* alla modalità *wireframed* e viceversa.
- L
  Accende/spegne le luci.
- M
  Passa dalla modalità di visualizzazione coi materiali, a quella senza. In modalità *wireframed* non si avverte differenza.
- Tab
  Passa da modalità oggetti alla modalità selezione vertici e viceversa.
- >
  Una volta in modalità *Vertex Edit* e assegnato un ruolo al modello selezionato, permette di scorrere in avanti il vettore dei punti d'assemblaggio caratteristici.

- <
  Una volta in modalità *Vertex Edit* e assegnato un ruolo al modello selezionato, permette di scorrere all'indietro il vettore dei punti d'assemblaggio caratteristici.

- +
  In modalità *Vertex Edit* permette di scorrere in avanti il vettore dei vertici che caratterizzano la geometria del modello selezionato.
  Permette inoltre di scegliere fra le varie normali se si sta introducendo manualmente un nuovo punto di assemblaggio.

- -
  In modalità *Vertex Edit* permette di scorrere all'indietro il vettore dei vertici che caratterizzano la geometria del modello selezionato.
  Permette inoltre di scegliere fra le varie normali se si sta introducendo manualmente un nuovo punto di assemblaggio.

- Enter
  Selezionando manualmente vincoli di assemblaggio permette di confermare, in caso di normali multiple, quella voluta.

- è
  Ruota la telecamera verso l'alto

- à
  Ruota la telecamera verso il basso

- ò
  Ruota la telecamera verso sinistra

- ù
  Ruota la telecamera verso destra

- Backspace
  Riporta la telecamera nella sua posizione iniziale

- X
  Elimina l'oggetto selezionato
- I
  Importazione di un nuovo modello
- E
  Esportazione del modello selezionato.

4.11 Funzionalità avanzata: Custom Shelf

Per simulare una progettazione procedurale di elementi d'arredamento, in questo caso, o di qualsiasi altro oggetto altamente modulare, si è implementata una funzionalità che permettesse di costruire in maniera rapida e intuitiva una scaffalatura, in cui fosse l'utente a decidere il numero di di scaffali e di mensole per ognuno di essi.

Il procedimento è molto semplice, in quanto sfrutta la metodologia implementata: esiste un componente di base, caratterizzato da diversi punti di assemblaggio, che viene importato all'interno della scena, duplicato tante volte quanto serve per completare la richiesta dell'utilizzatore. Lo pseudo-codice che realizza l'assemblaggio è molto semplice:

```
totComponenti = numScaffali * numMensole
importa il primo componente e sistemalo correttamente
for(i = 1 ; i < totComponenti ; i++)
{
    if( i % numScaffali == 0)
    {
     monta il primo modulo del piano superiore sul primo di quello inferiore
    }
    else
    {
     monta il modulo corrente alla sinistra del precedente
    }
}
```

Capitolo 5

Il programma in esecuzione

In questo capitolo presentiamo alcuni esempi d'utilizzo dell'applicativo. Descriveremo dall'importazione ed esportazione dei modelli, all'assemblaggio di alcuni di questi, dai più banali ai più elaborati, e ancora all'utilizzo della funzionalità avanzata *Custom Shelf* per la creazione automatica di scaffalature.

5.1 Esecuzioni di casi standard

In questo capitolo si prenderanno in esame alcune esecuzioni tipo, cercando allo stesso tempo di testare le prestazioni del *software* e di mostrarne il funzionamento complessivo. Si lavorerà con modelli molto semplici, a volte addirittura senza informazioni sul materiale, in quanto ci interessa più la parte relativa alle funzionalità del programma stesso, piuttosto che la modellazione precisa delle componenti.

5.2 Importazione di modelli e panoramica dell'ambiente

Il programma all'avvio si presenta con una finestra 3D già predisposta al lavoro, con una prima stanza preassemblata. L'utente sarà libero di modificare direttamente i modelli presenti nella scena, o eliminarli tutti e cominciare un nuovo lavoro.

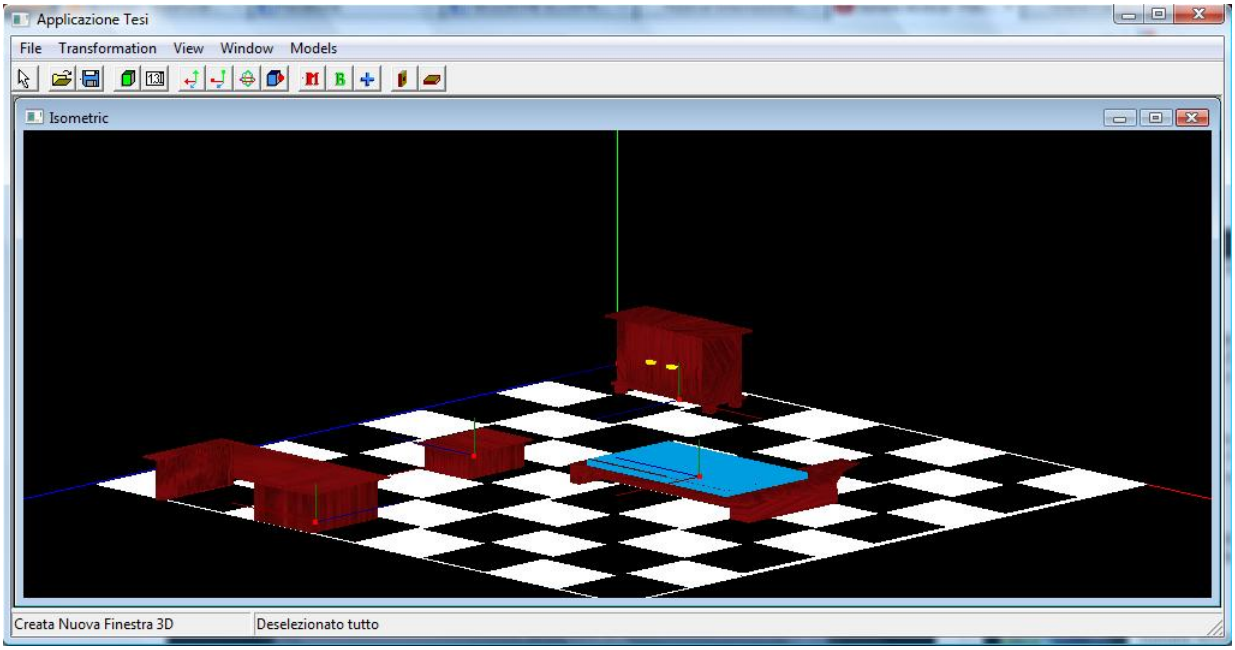

Figura 5.1: Il programma all'avvio

Per l'importazione di un modello, in formato OBJ o OBJ Esteso, fare *click* sul menù *Models>Import...* e, per mezzo della finestra di navigazione, selezionare il *file* desiderato. Se l'operazione è andata a buon fine l'oggetto verrà importato all'interno della scena, e il suo centro fatto coincidere con quello della scena stessa. Per selezionarlo, e poter applicargli trasformazioni o assemblaggi, sarà sufficiente fare *click* su di esso col tasto sinistro del mouse.

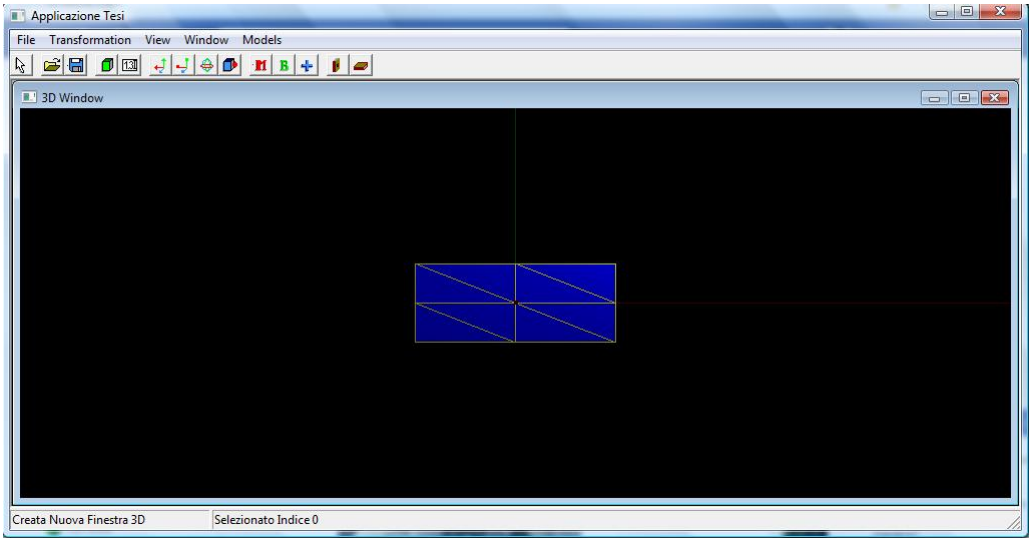

Figura 5.2: Un modello importato e selezionato

Per mezzo della voce di menù *File>New* (o dell'apposito bottone nella *toolbar*) sarà possibile aprire più viste sulla scena, ognuna in una finestra indipendente. Si potrà modificare la visuale all'interno di ognuna di esse sia attraverso le apposite voci di menù, che attraverso i tasti è, à, ò, ù.

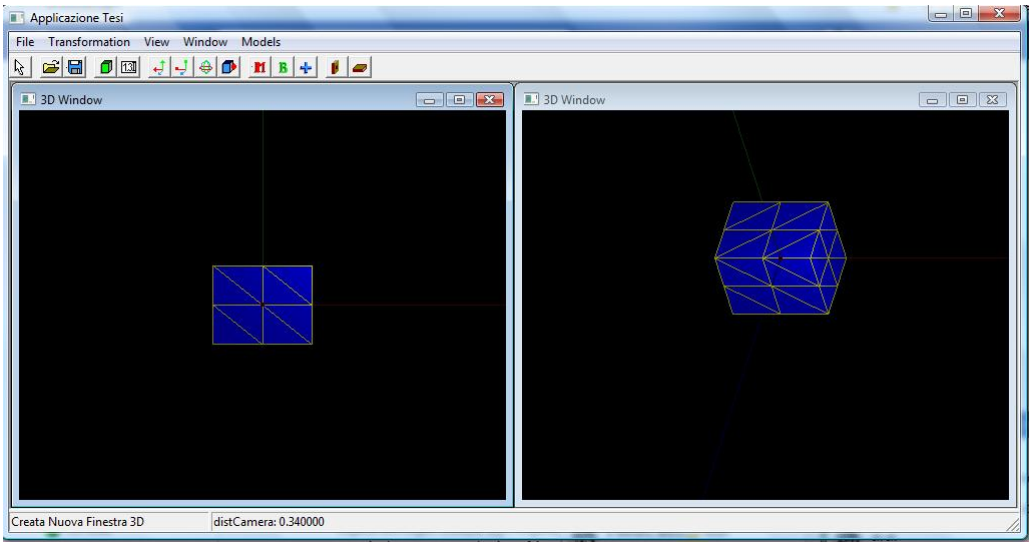

Figura 5.3: Più viste sul modello

5.3 Inserimento di *Assembly Group*

Per far sì che gli assemblaggi siano possibili occorre, importando un modello OBJ standard come quello che si sta usando per il nostro esempio, individuare su di esso tutti gli *Assembly Group* che possono poi essere utili durente le operazioni di *mating*. Per farlo occorre seguire i passi seguenti:

- Selezionare il modello: *click* col tasto sinistro del mouse, un modello è selezionato quando evidenziato in giallo

- Passare in modalità *Vertex Edit*: per mezzo della voce di menù *Models>Vertex Edit* o con il tasto Tab si passa alla visualizzazione dei vertici del modello

- Selezionare il vertice: si seleziona con un *click* col tasto sinistro del mouse, si possono scorrere i vertici anche per mezzo dei tasti + e -

- Avviare la procedura di salvataggio del gruppo di assemblaggio: selezionare la voce di menù Models>Vertices>Add Assembly Group

- Selezione Normale *Fits*: con i tasti + e - scorrere il vettore delle normali, confermando la normale individuante il vincolo *Fits* premendo Invio

- Selezione Normale *Against*: con i tasti + e - scorrere il vettore delle normali, confermando la normale individuante il vincolo *Against* premendo Invio

- Tornare in modalità selezione: premere nuovamente il tasto Tab per uscire dalla modalità *Vertex Edit*

Nota: la procedura permette di inserire solo vincoli di tipo Fits ed Against. Questo non è un problema per i nostri scopi, in quanto ci era sufficiente mostrare un funzionamento tipo per una procedura di inserimento vincoli, ed inoltre con i due soli vincoli implementati si possono risolvere la maggior parte dei casi con cui ci dovrà confrontare

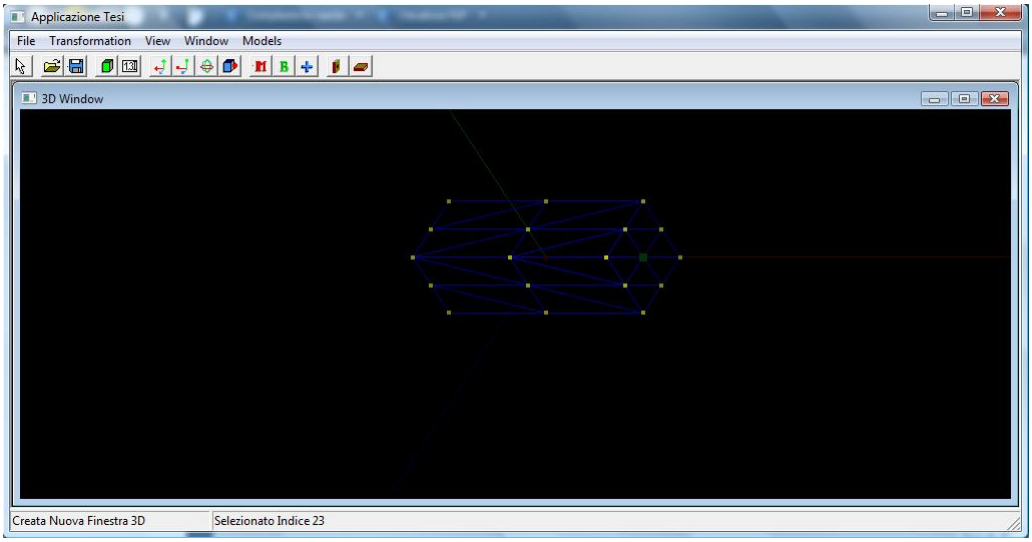

Figura 5.4: Modello in modalità Vertex Edit

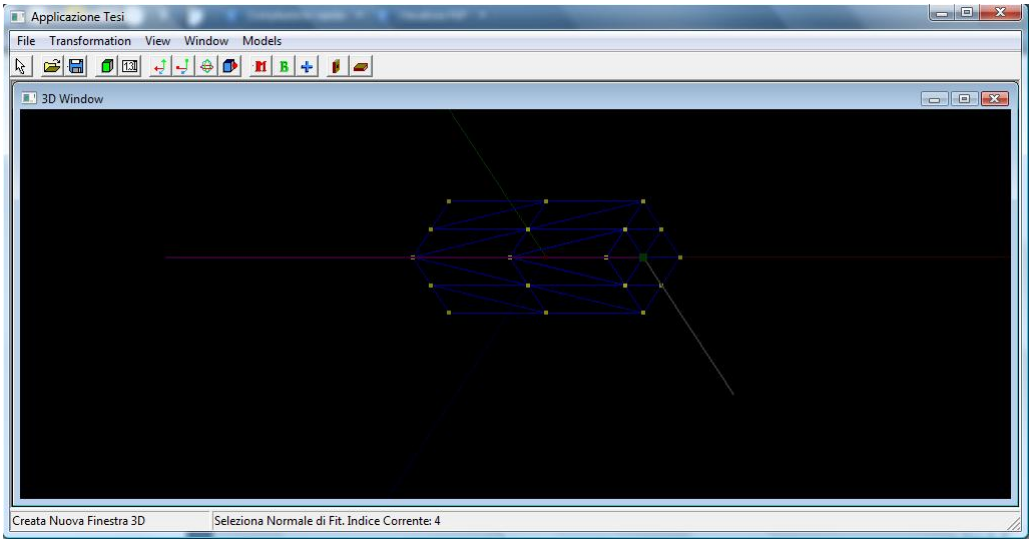

Figura 5.5: Selezioni delle normali per l'*Assembly Group*

5.4 Esportazione del modello

Una volta inseriti i gruppi d'assemblaggio necessari è possibile esportare il modello in formato OBJ Esteso, in modo tale che sia possibile importare copie dello stesso

con vincoli per la fase di *mating* già correttamente impostate. Ogni oggetto tridimensionale importato memorizza il percorso al *file* che lo descrive; lo si recupera e se ne copia il contenuto all'interno di un nuovo *file*, aggiungendo le informazioni dei gruppi d'assemblaggio. Riportando solo la parte legata alla geometria del modello, è possibile modificare anche quegli oggetti che sono già espressi in formato OBJ Esteso.

5.5 Primi Assemblaggi

Siamo ora in possesso di un modello correttamente descritto, per quel che riguarda assemblaggi futuri, da vincoli consistenti di *mating*. Operiamo ora un assemblaggio fra due istanze del modello su cui abbiamo lavorato finora, mostrando la due modalità di lavoro possibili, quella manuale e quella automatica.

5.5.1 Modalità Manuale

Per tale modalità occorre selezionare due modelli, uno per volta, ed assegnar loro un *ruolo*. Per portare a termine correttamente un assemblaggio occorre individuare un modello *Mate* ed un modello *Base*. Fatto questo, grazie alle voci di menù *Models>Role>Mate Component* e *Models>Role>Base Component*, si può scegliere in modalità *Vertex Edit*, quale *Assembly Group* utilizzare per ognuno dei due componenti. Per operare tale scelta occorre

- Selezionare un modello
- Entrare in modalità *Vertex Edit*
- Utilizzare < e > per navigare il vettore di *Assembly Group*
- Tornare in modalità Selezione per confermare la scelta

Con tutte le impostazioni correttamente impostate, si può avviare l'assemblaggio vero e proprio dal menù Models>Assembly o tramite il bottone apposito nella *toolbar*.

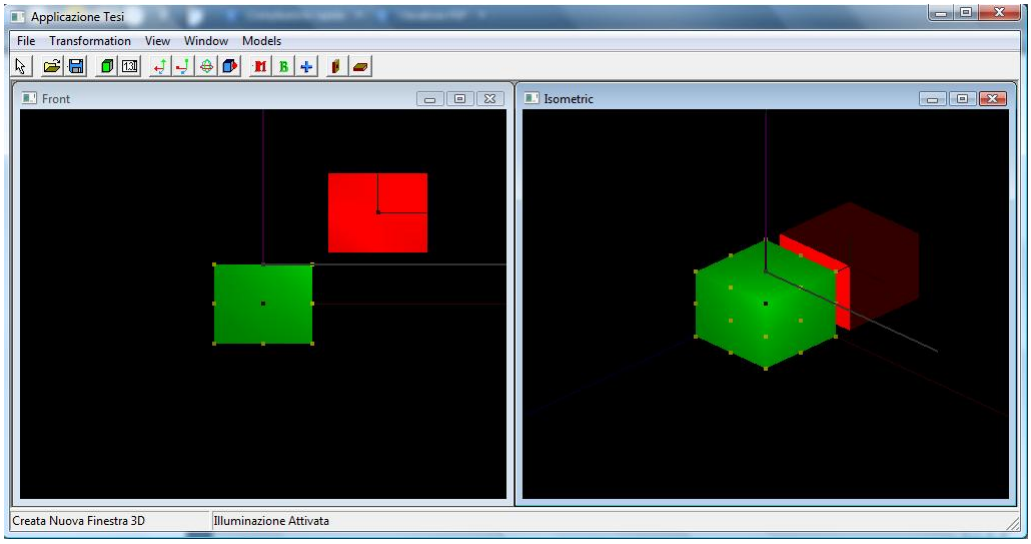

Figura 5.6: Modelli Base (Selezionato) e Mate

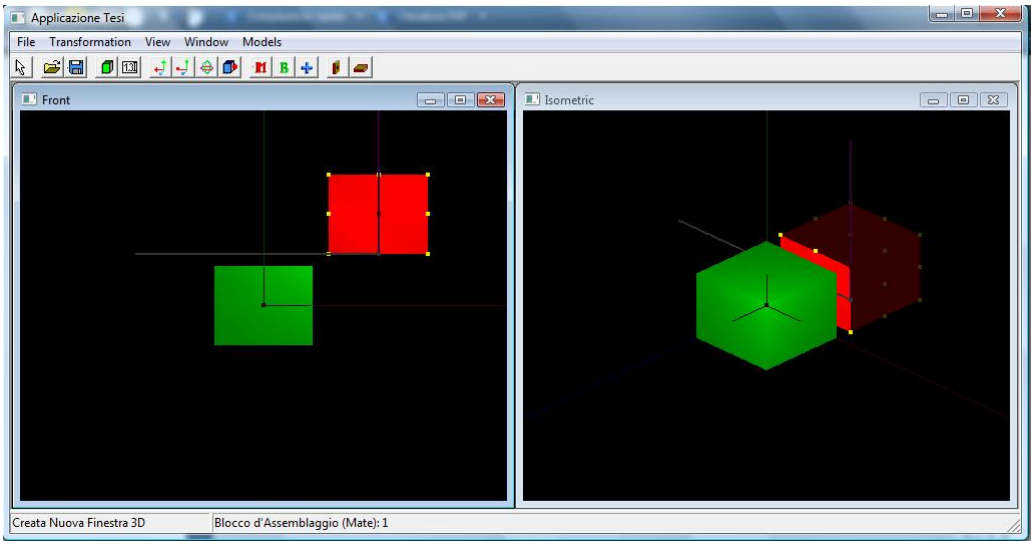

Figura 5.7: Modelli Base e Mate (Selezionato)

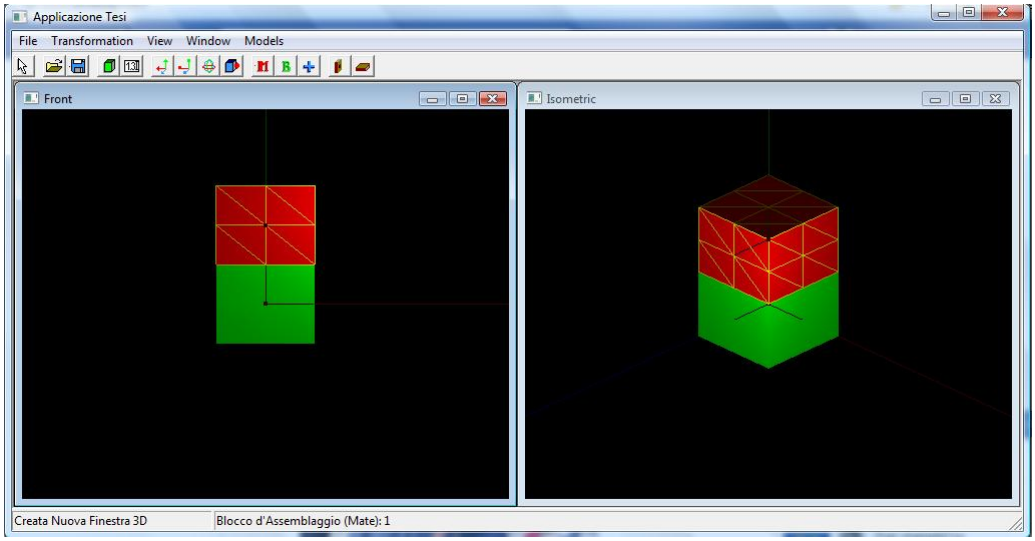

Figura 5.8: Un assemblaggio

Notiamo che il risultato è esattamente quello voluto, con il componente *Mate* (in rosso) esattamente sopra al componente *Base* (in verde).

5.5.2 Modalità Interattiva

In questa modalità gli assemblaggi risultano incredibilmente intuitivi. Tutto quello che l'utente deve fare è selezionare il modello che vuole assemblare e avvicinarlo, tenendo premuto il tasto sinistro del mouse, al modello su cui vuole applicare l'operazione di *mating*. Rilasciando l'oggetto, si individuerà automaticamente come componente *base* il modello più vicino a quello manipolato, e i gruppi d'assemblaggio che prenderanno parte all'operazione stessa saranno scelti sulla base della minor distanza reciproca. L'utente potrà rendersi conto di essere in questa modalità di assemblaggio interattivo, in quanto il cursore del mouse prenderà le fattezze di una piccola chiave inglese.

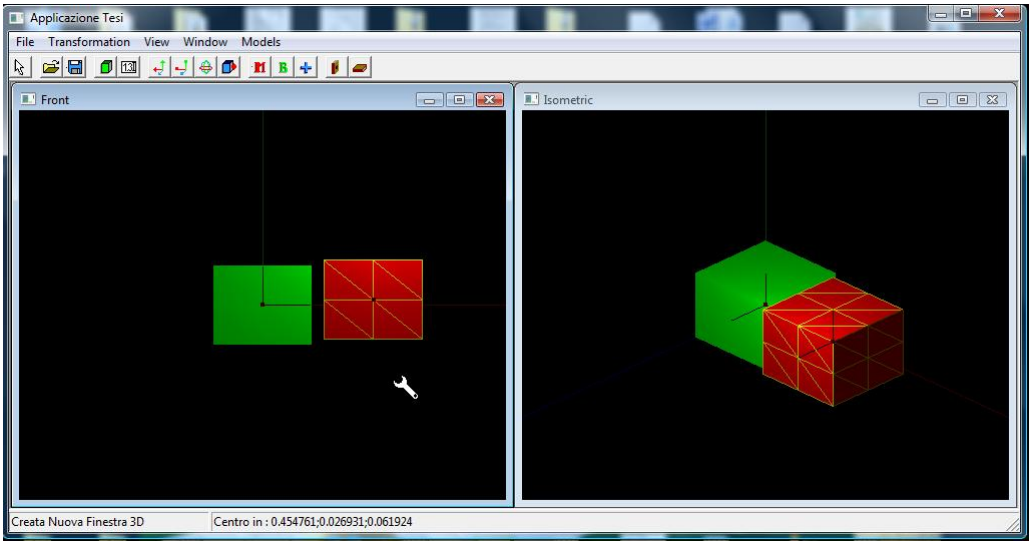

Figura 5.9: Un assemblaggio interattivo in esecuzione...

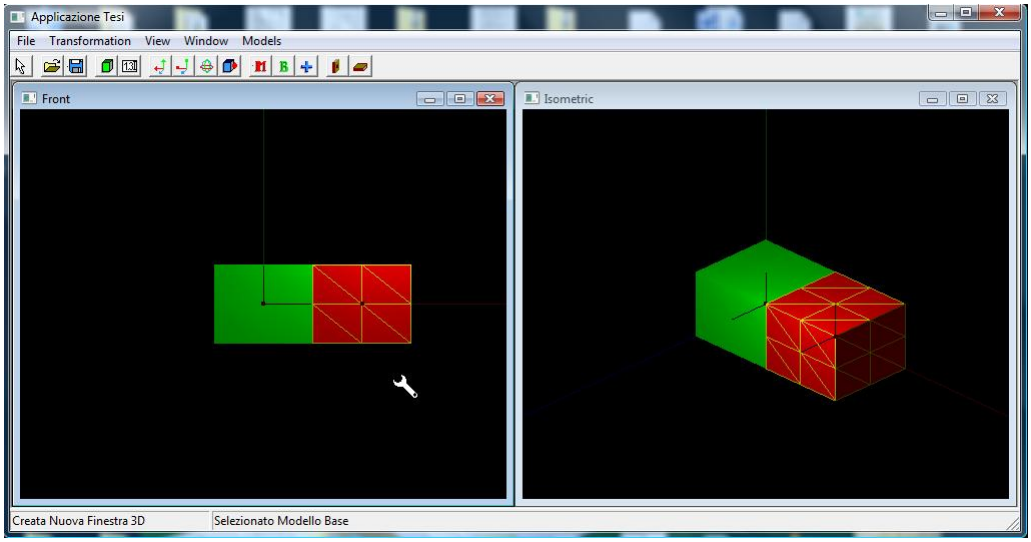

Figura 5.10: ...e il suo risultato

5.6 Custom Shelf

Questa modalità è stata inserita per simulare esigenze di creazione di grandi oggetti complessi altamente modulari. Nel nostro caso abbiamo ipotizzato di dover progetta-

re una libreria, in cui il numero di scaffali e il numero di mensole per scaffale fossero un parametro inserito dall'utente. Si avvia la modalità attravero la voce di menù *Models>Custom* Shelf. Comparirà la finestra di dialogo in figura 4.4; inserendo i dati numerici e premendo successivamente il tasto Crea si avvierà la procedura.

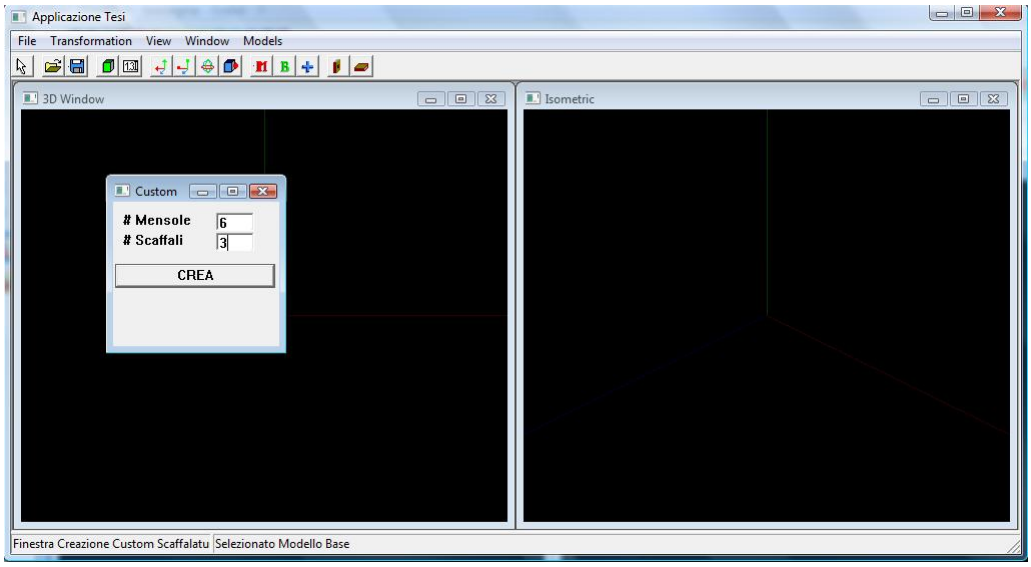

Figura 5.11: L'inserimento dei parametri

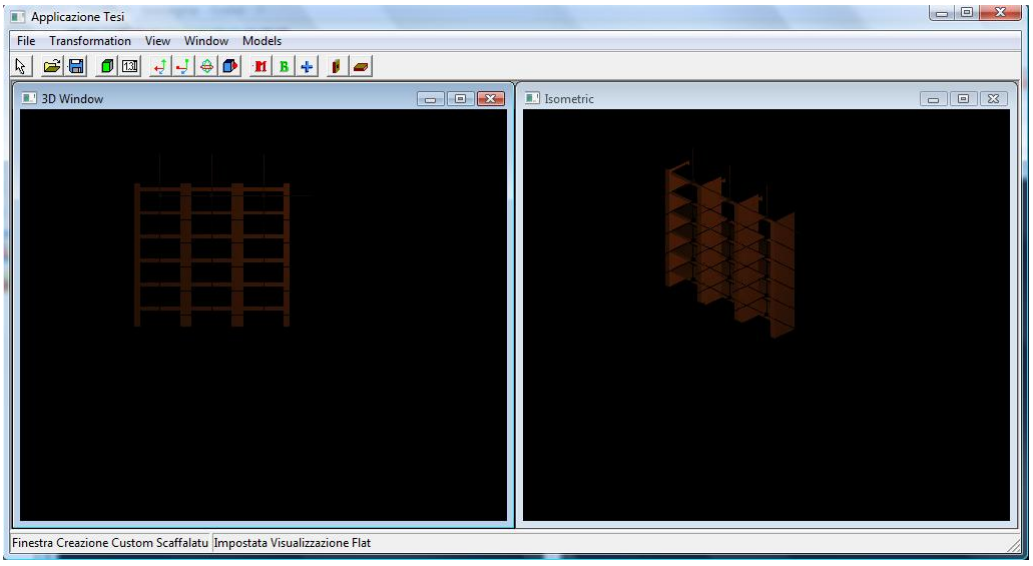

Figura 5.12: Il risultato dell'operazione

5.7 Una piccola volumetria di una cucina

Supponendo che l'applicazione venga utilizzata da un mobilificio, abbiamo modellato dei semplici modelli a simulazione di componibili per una cucina: un frigorifero, un mobiletto basso e una sorta di forno. Importati nella nostra scena, per mezzo della funzionalità Assemblaggio Interattivo possiamo disporli a creare varie soluzioni d'arredamento, che soddisfano le esigenze di un cliente fittizio.

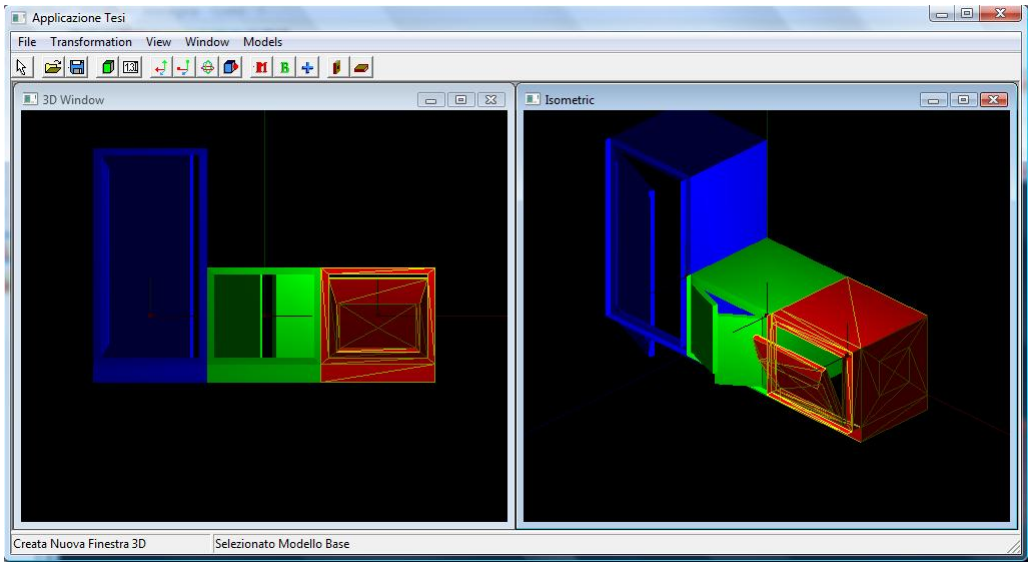

Figura 5.13: Una prima proposta...

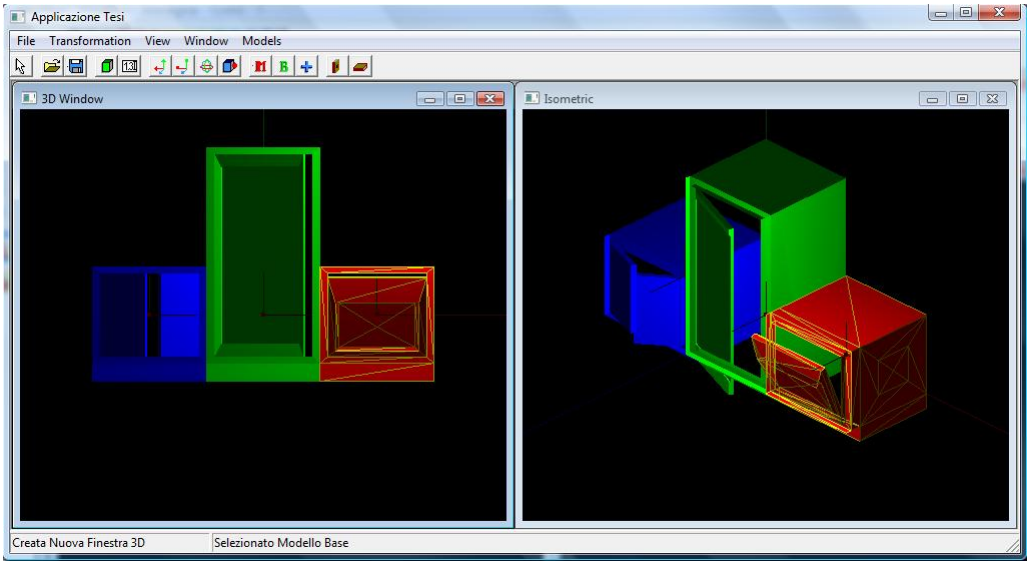

Figura 5.14: ...una seconda...

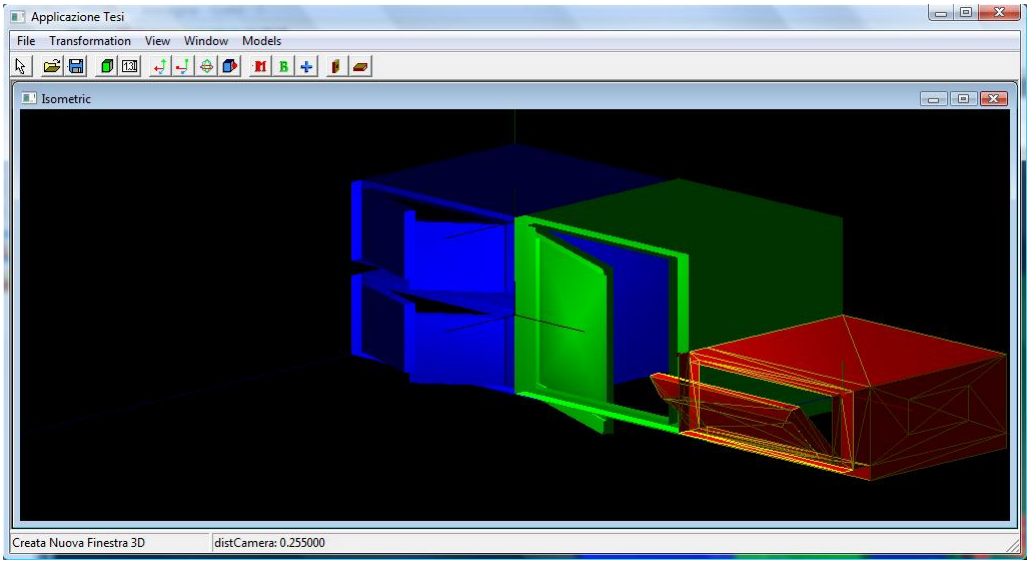

Figura 5.15: ...e una terza

5.8 Altre volumetrie

Negli esempi precedenti abbiamo supposto che la nostra applicazione fosse fortemente mirata a soddisfare le necessità di un mobilificio; mantenendo valida questa ipotesi mostriamo altre possibili soluzioni ottenute utilizzando l'applicativo.

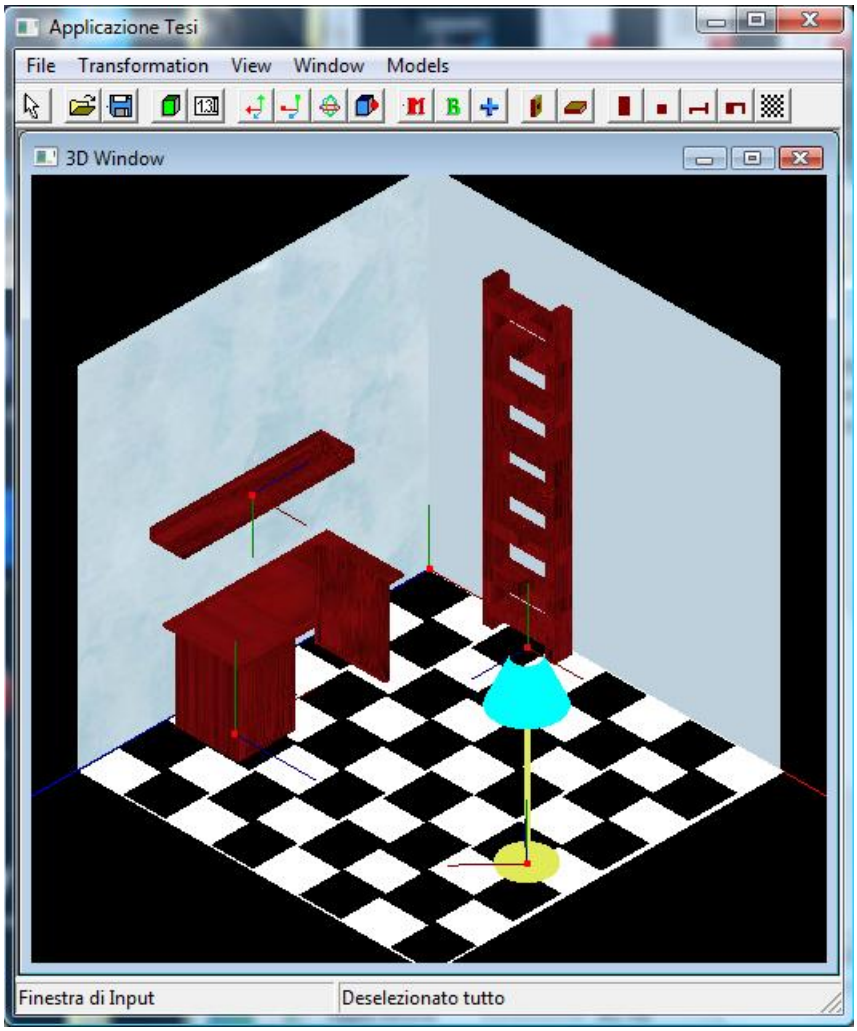

Figura 5.16: Una seconda stanza

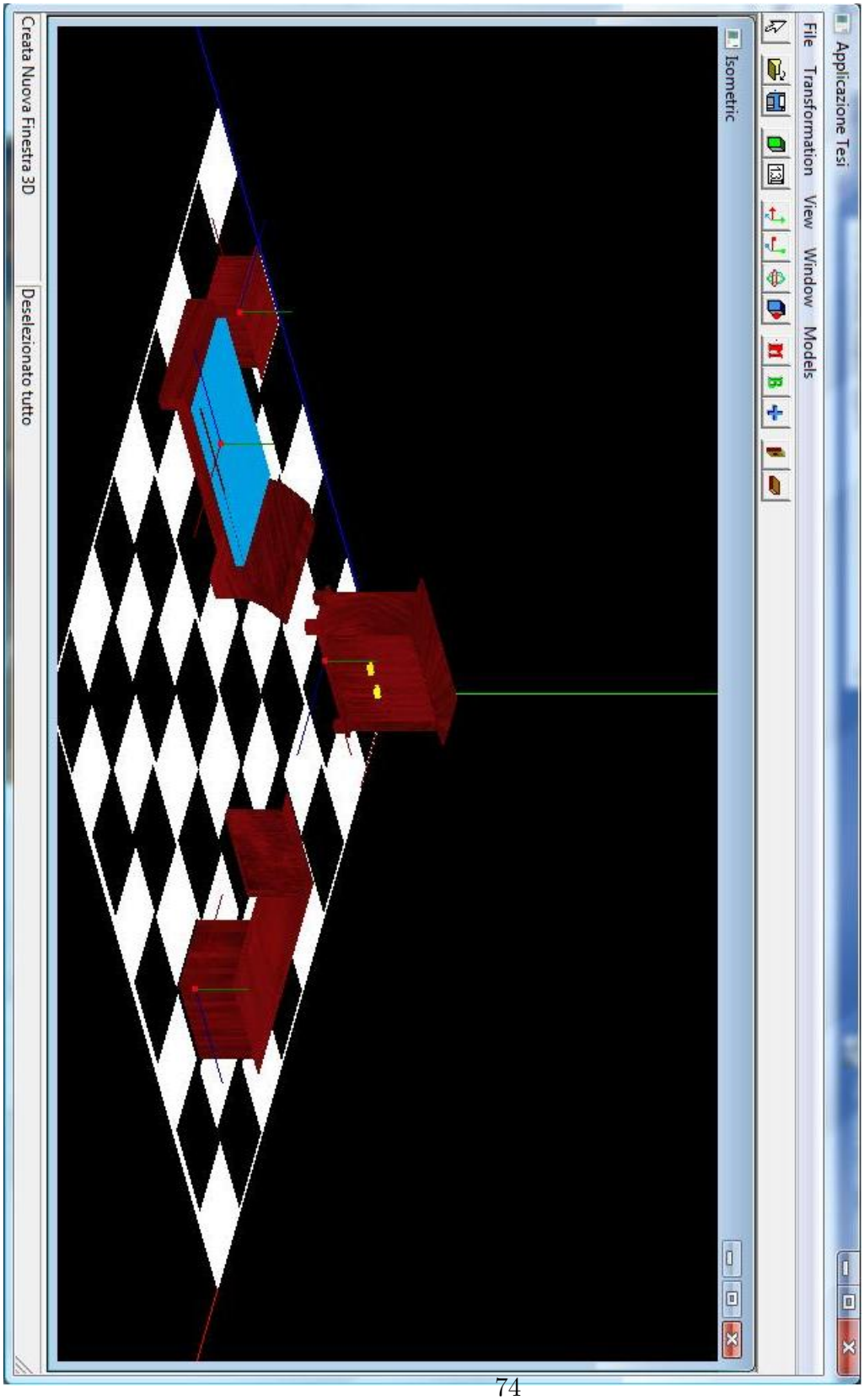

Figura 5.17: Una seconda proposta per la camera iniziale

Conclusioni

Abbiamo implementato e illustrato il funzionamento di un *software* CAD tridimensionale, fortemente mirato alla risoluzione di problematiche che possono essere proprie di un mobilificio. L'ambito applicativo non è vincolante, con poche modifiche si potrebbe agevolmente cambiare il *target* di mercato. Il concetto implementato è altamente astratto ed applicabile a qualsiasi situazione preveda assemblaggi di parti semplici per ottenere un nuovo oggetto complesso.

Abbiamo valutato alcuni aspetti che rendono un CAD come quello prodotto uno strumento ottimale per un'azienda produttrice di qualsiasi sorta di bene. Rende infatti possibile una progettazione rapida, che non necessita della produzione di prototipi: queste due caratteristiche combinate si traducono in un risparmio immediato per l'azienda, sia in termini di tempo che di denaro.

Tutto l'applicativo si basa su tecnologie aperte o quantomeno gratuite: Blender è utilizzato per la fase di creazione dei modelli di base, Dev C++ è stato l'IDE per lo sviluppo del codice, mentre per gestire la parte grafica ci si è serviti delle librerie OpenGL. Questo permette di ipotizzare il futuro del codice in una duplice ottica: esso potrebbe essere rilasciato in maniera aperta, in modo da essere migliorato e modificato dai singoli utilizzatori finali, o semplicemente potrebbe essere venduto ad un prezzo altamente competitivo, in quanto nessuna spesa è stata sostenuta per gli strumenti di sviluppo.

La problematica trattata è ovviamente ampia, e nuove esigenze potrebbero emergere. Il programma potrebbe subire una numerosa serie di migliorie, come per esempio la possibilità di leggere nuovi formati di *file* che possano importare una stanza ammobiliata in precedenza. Ancora si potrebbe pensare di integrare il sistema con un semplice *editor* di immagini, in modo da modificare sul momento le *texture* dei singoli modelli per lascia-

re la possibilità all'utente di valutare quale tipologia di materiale restituisca il miglior impatto visivo secondo il suo personale giudizio.

Bibliografia

[1] J. Kim, K. Kim, K. Choi and J. Y. Lee, *Solving 3D Geometric Constraints for Assembly Modelling*, 2000.

[2] A. P. Ambler and R. J. Popplestone, *Inferring the position of bodies from specified spatial relationships*, Artificial Intelligence, 6, pp. 157−174, 1975.

[3] D. Rocheleau and K. Lee, *System for interactive assembly modeling*, Computer-Aided Design, 19(2), pp. 6572, 1987.

[4] F. Tomas and C. Torras, *A group-theoretic approach to the computation of symbolic part relations*, IEEE Transactions on Robotics and Automation, 4(6), pp. 622634, 1988.

[5] L. S. Haynes and G. H. Morris, *A formal approach to specifying assembly operations*, International Journal of Machine Tools and Manufacture, 28(3), pp. 281298, 1988.

[6] G. A. Kramer, *Solving geometric constraint system: A case study in kinematics*, MIT Press, 1992.

[7] J. U. Turner, S. Subramaniam and S. Gupta, *Constraint representation and reduction in assembly modeling and analysis*, IEEE Transactions on Robotics and Automation, 8(6), pp. 741750, 1992.

[8] R. Anantha, G. A. Kramer and R. H. Crawford, *Assembly modeling by geometric constraint satisfaction*, Computer-Aided Design, 28(9), pp. 707722, 1996.

[9] V. N. Rajan, K. W. Lyons and R Sreerangam, *Generation of component degrees of freedom from assembly surface mating constraints*, Proceedings of ASME Design Engineering Technical Conference, DETC97/DTM-3894, September 1997.

[10] A. Muolo, *Potenzialità dei Sistemi CAD per la flessibilità dei Sistemi Produttivi*, 2011.

[11] D. Astle, K. Hawkins, *Beginning OpenGL: Game Programming*, 2008/09

[12] D. Lazzaro, *Lezioni di Metodi Numerici per la Grafica*, 2011/12.

Sitografia

[OpenGL] http://www.opengl.org/

[OpenGL, References] http://www.opengl.org/sdk/docs/man/

[WINAPI, Refereces] http://msdn.microsoft.com/en-us/library/windows/desktop/ms632586

[WAVEFRONT(OBJ), Definition] http://www.martinreddy.net/gfx/3d/OBJ.spec

[Blender] http://www.blender.org

[Blender, Storia] http://www.blender.org/blenderorg/blender-foundation/history/