ALMA MASTER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Specialistica in Informatica

---

Tesi di Laurea

# Progettazione e Sviluppo di una Tecnica di Rendering di Oggetti 3D in Scene Reali

Relatore:
Chiar.mo Prof. Giulio Casciola

Candidato:
Stefano Bertozzi

Correlatore:
Ing. Giovanni Panzieri

---

Anno Accademico 2010-2011
Sessione III

# Indice

# Elenco delle figure

# Introduzione

## Contesto

La resa di immagini sintetiche ottenute da scene virtuali è oggi una tecnica molto utilizzata in tutti i settori che adoperano software 3D. Con "resa" si intende quel processo che crea una immagine 2D della scena geometrica 3D, solitamente illuminata tramite luci sintetiche, dal punto di vista e nella direzione di vista della camera.
In particolare, aziende che utilizzano software di arredamento, architetti, designer, arredatori, fotografi, fanno un pesante uso di queste rese per ottenere dei risultati visivi di ottimo livello; risultati qualitativamente così buoni che spesso queste immagini sono indistinguibili da vere fotografie. Tecniche sempre più innovative vengono introdotte continuamente nel settore della resa di scene 3D per avere risultati sempre migliori e prossimi al reale.
La modellazione del background ( ovvero tutto quello che appare dietro il nostro punto di interesse nella immagine ottenuta, come panorami, pareti ed elementi di interni, ecc. . . ) e la corretta illuminazione di scene ed oggetti, rappresentano una parte molto lunga e tediosa e non sempre è possibile o facile ottenere risultati realistici. Sarebbe molto più veloce, performante e preciso uno strumento che permetta di evitare questi due passi, lasciando all'utilizzatore solo il compito di modellare gli oggetti veramente importanti, che siano il punto focale della scena.

Una tecnica recente e molto promettente prevede di utilizzare delle foto-

grafie direttamente inserite nel contesto 3D, sovrapponendo ad essa oggetti sintetici; così facendo la prospettiva inganna l'occhio umano e gli oggetti sembrano realmente presenti nello scatto fotografico. Inoltre, sempre da foto inserite nel contesto, è possibile estrapolare, tramite varie tecniche, il contributo di luce della scena, permettendo così un'illuminazione ottima degli oggetti sintetici.

## Motivazione

Questa tesi è stata sviluppata presso l'azienda *Computer Office s.r.l.* che si occupa di sviluppo e gestione di un software proprietario di arredamento 3D; software che trova mercato in grandi e piccoli nomi dell'industria del mobile e dell'arredamento di interni.
L'azienda ha voluto esplorare una nuova metodologia di creazione di una resa fotorealistica alternativa alla classica metodologia del *ray tracing*, incaricandomi di questo progetto di ricerca e sviluppo. Questo lavoro avrebbe sicuramente accresciuto il know-how aziendale e posto le basi per creare, magari dopo accorti ritocchi e modifiche, una "ramificazione" del software base, così da proporlo ai clienti come tecnologia innovativa e performante ed ottenere un vantaggio rispetto alle aziende concorrenti.

## Contributo ed Obiettivi finali

Questa tesi vuole esplorare un settore in fase di sviluppo nel campo della Computer Graphics, quello dell'utilizzo di fotografie in un contesto 3D. Proporre quindi una tecnica alternativa a quelle già presenti, valutandone la bontà dei risultati.

L'obiettivo finale era quello di realizzare, dopo attento studio e progettazione, una tecnica di render di un contesto 3D ( in questo caso mobilia ) correttamente posizionato su uno sfondo reale ( una fotografia digitale di interni non arredati ) e correttamente illuminato dal contributo diffuso delle luci presenti nella stanza fotografata ( che non utilizzasse quindi una illuminazione artificiale creata attraverso il motore grafico ), tramite strumenti open-source o a disposizione dell'azienda, tutto correttamente integrato in un unico progetto in modo da poter essere successivamente inserito facilmente nel software proprietario. Il tutto "user-friendly" e "tollerante", per agevolare il lavoro dei clienti.

## Overview dei capitoli successivi

Nel primo capitolo verrà presentato e spiegato concettualmente l'algoritmo elaborato, ottenuto dalla fase di progettazione. Nel secondo capitolo andremo a vedere l'ambiente di sviluppo, le tecnologie e le librerie scelte per lo sviluppo del codice, motivandone anche le scelte. Nel terzo capitolo saranno spiegate e mostrate le interfacce create per l'utilizzo del software; interfacce di raccolta dati e gestione eventi che portano avanti per passi successivi l'algoritmo, il cui sviluppo e dettagli sono presentati nel capitolo quattro. Il successivo capitolo presenta i risultati, evidenziando pro e contro delle tecniche usate. Infine il capitolo sei rivede il lavoro presentato, proponendo sviluppi futuri e traendo conclusioni su quanto fatto.

# Capitolo 1

# Progettazione

## 1.1 Premesse

La progettazione dell'algoritmo parte da un presupposto base: un cliente, avendo intenzione di arredare una o più stanze di casa propria, si presenta ad un rivenditore con una o più foto della stanza d'interesse. Quello che si vuole ottenere è la foto stessa della stanza in varie versioni, dove ognuna presenta una mobilia, tra quelle disponibili del rivenditore, inserita ed illuminata correttamente, così che il cliente possa effettivamente vedere la sua stanza arredata. Come già spiegato, l'utilizzo di una foto come sfondo permette di velocizzare notevolmente il processo ( si evita completamente la fase di modellazione poichè ovviamente i mobili del rivenditore sono già creati e disponibili, ma soprattutto poichè la stanza è già presente come sfondo e non richiede di essere modellata nè illuminata ) e di ottenere risultati più realistici.

Inoltre sono stati posti alcuni vincoli legati ai tool di sviluppo, come l'utilizzo di *Visual Studio* e del linguaggio *C++*, l'uso esclusivo di librerie free che si potessero integrare nel codice, senza quindi ricorrere a strumenti esterni.
Necessaria era una buona usabilità e portabilità del progetto ( interfacce facili e chiare; processi veloci; dinamicità del progetto, soprattutto tramite path dinamici in modo da non avere vincoli sul percorso del progetto ).

Infine, eventuali ulteriori richieste di dati al cliente sarebbero dovute essere "fattibili" per una persona comune; ad esempio la richiesta di eventuali ulteriori scatti fotografici dell'interno non poteva prevedere nulla di più complesso di scatti semplici.

## 1.2 Algoritmo

L'algoritmo prevede, una volta creato un contesto 3D in cui sono inseriti i mobili sotto forma di mesh, con relativi materiali e texture, di orientare la camera 3D secondo il punto di vista della fotocamera per poi posizionare la fotografia come sfondo. In questa maniera i mobili risultano correttamente posizionati, grazie alla prospettiva, nella fotografia. Per svolgere questa operazione di *Camera Calibration o Camera Resectioning* ( si veda [wiki1] ) si è deciso di adottare una tecnica, spiegata in dettaglio nei prossimi capitoli, che calcola la matrice di proiezione e la matrice mondo a partire da 5 o più punti, detti di controllo, in coordinate mondo e dalle coordinate dei punti corrispettivi nella vista nella fotografia, ottenendo così dei parametri camera adatti allo scopo.

Successivamente sarà necessario illuminare le mesh in maniera conforme alla fotografia. Utilizzando la tecnica di illuminazione basata su immagini tramite un *Cube Mapping* ( tecnica di texturing che ricade nella categoria dell'*Environment mapping*, spiegato in sezione 2.2.6 ), è possibile far sì che ogni oggetto texturato tramite questa tecnica rifletta perfettamente l'ambiente che lo circonda. Da questa texture si è pensato poi di estrapolare il solo contributo diffuso della luce, tale da garantire alle mesh una illuminazione adeguata; le fotografie di un ambiente sono infatti una rappresentazione delle varie luci presenti nell'ambiente stesso e texturare un oggetto con queste foto corrisponde con una buona approssimazione ad illuminarlo con le luci stesse. Inoltre la luce può essere scomposta in diversi contributi, tra cui quello dif-

fuso che rispecchia l'illuminazione globale, ambientale, ovvero quella che non deriva direttamente da fonti di luce ma da varie riflessioni ed attenuazioni di una o più fonti ( ad esempio si pensi al chiarore del cielo quando il sole non è visibile a causa di nubi ).
In prima analisi si è ritenuto sufficiente ricavare questo contributo di luce dalla singola fotografia, approssimando la stanza con un cubo su cui applicare, come texture, le porzioni di fotografia che rappresentano le pareti, ottenendo così un Cube Map da utilizzare per l'illuminazione delle mesh. Questo però si è visto essere non sufficiente a generare un Cube Map completo, essendo presenti nelle fotografie due o tre pareti al massimo.
In letteratura esistono tecniche che utilizzano strutture avanzate per ottimizzare questa procedura; sono chiamate *Light Probe*, ovvero fotografie a 360° utilizzabili direttamente come Environment Map, ricavate da varie fotografie fatte su una sfera riflettente posta al centro della scena e successivamente combinate tramite software di elaborazioni di immagini per eliminare le distorsioni.
Ovviamente per noi era impensabile utilizzare tali strutture a causa delle premesse fatte. Si è deciso perciò di richiedere al cliente degli scatti aggiuntivi: fotografie ortogonali alle pareti, al soffitto ed al pavimento, per un totale di 6 scatti, posizionando la camera all'incirca nel punto della stanza in cui andrà messa la mobilia. Questi scatti sono sicuramente alla portata di chiunque, anche se non ritraggono perfettamente l'intera stanza. Il processo non risulta sicuramente esatto, ma anzi, un po' grossolano; si è ritenuto comunque sufficiente utilizzare il Cube Map così creato per garantire una illuminazione diffusa alle mesh, componente che viene ricavata tramite l'applicazione di un filtro di convoluzione ( sezione 2.2.7 ) alle foto stesse. L'utilizzo della sola componente diffusa fornisce un buon margine di tollerabilità sulla precisione delle foto, al contrario della riflessione. Inoltre l'obiettivo finale non era quello di ottenere un risultato preciso, soprattutto dal punto di vista fisico e matematico, ma una immagine che risultasse gradevole all'occhio.

Infine abbiamo previsto l'utilizzo della tecnica di *Tone Mapping*, tecnica usata per mappare un set di colori in un altro, in modo da approssimare l'aspetto finale della scena ad uno scatto HDRI, ovvero una immagine con un più ampio, rispetto al normale, range di valori compresi tra i picchi massimo e minimo di luminosità, per ottenere una rappresentazione più accurata dei livelli di luminosità della scena; questo metodo risulta utile soprattutto in presenza di forti contrasti di luce. Esso sfrutta il pixel shader della scheda grafica ( una delle componenti delle attuali schede grafiche programmabili; effettua elaborazioni sui singoli pixel ), consentendo di migliorare la luminosità della scena e soprattutto la colorazione finale delle mesh, miscelandone, secondo nostre istruzioni, il colore base, eventuali texture applicate ad esse ( come il disegno del tessuto di un divano ) ed il contributo diffuso ottenuto dal processo di Cube Mapping.

Ricapitolando:

1. Creazione o caricamento mobilia 3D;
2. Processo di Camera Calibration;
3. Gestione dell'illuminazione della mobilia tramite foto dell'ambiente;
4. Correzioni di colore e luminosità per una resa migliore.

I passi dettagliati di questo algoritmo, ottenuti tramite raffinamenti successivi, sono riportati nei capitoli successivi.

# Capitolo 2

# Ambiente di Sviluppo

Di seguito verrà illustrato l'ambiente di sviluppo dell'algoritmo; si parlerà di hardware utilizzato, poichè la resa finale di immagini ottenute da software di grafica 3D dipende anche dall'hardware che si utilizza ed in particolare dalla scheda grafica. Inoltre, sulla base delle scelte fatte in fase di progettazione, verranno indicate le librerie e quant'altro usato per lo sviluppo del codice.

## 2.1 Hardware

Mi è stato consentito dall'azienda l'utilizzo di una postazione con un pc con le seguenti caratteristiche:

- Windows XP
- RAM 1Gb
- processore Intel 2.8 Ghz
- scheda grafica NVIDIA 7100 GS
- monitor CRT utilizzato a risoluzione 1024x768

Le immagini qui riportate sono state catturate su un pc più performante:

- Windows 7
- RAM 6Gb
- processore Intel i7-720QM
- scheda grafica NVIDIA GeForce GTS 360M
- monitor LED utilizzato a risoluzione 1366x768

## 2.2 Tecnologie e Librerie

Di seguito quello che riguarda l'IDE, le librerie, le tecniche usate.

### 2.2.1 Ambiente di Sviluppo - IDE e Linguaggio di Programmazione

Tutto il lavoro è stato svolto tramite l'IDE *Visual Studio 2008 SP1*; questo è lo strumento base utilizzato dall'azienda per lo sviluppo software, solitamente tramite linguaggi derivati dal C. Nel mio caso si è voluto programmare in *C++* ( linguaggio ad oggetti ) per lavorare tramite classi che rappresentano ottimamente le varie parti dell'algoritmo, via via identificate tramite raffinamenti successivi. La programmazione ad oggetti e gli strumenti forniti da questo IDE, come il debugger, mi hanno fornito ottimi strumenti per gestire al meglio il lavoro.

### 2.2.2 Libreria d'interfaccia

La scelta è ricaduta immediatamente sulla libreria *MFC - Microsoft Fundation Class*, integrata nativamente nell'IDE scelto. Questa libreria permette di creare delle interfacce che abbiamo ritenuto più che sufficienti al nostro fine; inoltre consente di gestire eventi mouse e tastiera. Il primo approccio non è

risultato semplice e lo strumento non sembra, a mio avviso, snello e veloce; mai però avevo progettato e sviluppato interfacce se non tramite HTML e simili. Il risultato finale è stato comunque, come vedremo nei prossimi capitoli, soddisfacente per un progetto di ricerca.

### 2.2.3 Motore Grafico 3D

*Ogre - Object-Oriented Graphics Rendering Engine* è stato scelto per creare e gestire il contesto 3D. Ogre è un motore grafico 3D "scene-oriented" flessibile, scritto in linguaggio C++ e pensato per rendere facile ed intuitiva agli sviluppatori la produzione di applicazioni che utilizzano una grafica 3D con accelerazione hardware. Oltre ad essere progettato per mantenere determinate caratteristiche qualitative ed essere flessibile, Ogre ha una grande comunità web che permette di trovare sempre risposta ad interrogativi, problemi, ecc...; è anche corredato di una ottima documentazione, di molti tutorial, manuali, "how to". Infine, caratteristica necessaria, Ogre è disponibile free sotto licenza dell'MIT.
Dopo un largo utilizzo la mia opinione è che Ogre fornisce un ottimo strumento per la grafica 3D ed è capace di grandi cose. La sua bontà è però anche proporzionale alla sua difficoltà di apprendimento ed utilizzo; esso richiede infatti molto studio e pratica anche solo per utilizzare le funzioni base.

### 2.2.4 Libreria per lettura di mesh in formato 3DS

Il software aziendale esporta le mesh in formato 3DS; ho dovuto perciò integrare al codice la libreria fornitami dall'azienda, scritta in C++, per importare quanto creato in fase di design tramite *Giotto MobilCAD*, ovvero le pareti e i mobili. Successivamente è stato poi necessario convertire tutto quanto in strutture e dati utilizzabili da Ogre; creare tutti i passi necessari a ricreare, vertice per vertice, faccia per faccia, materiali, texture, ecc..., le mesh importate. Ogre legge infatti le mesh in un formato proprietario *.mesh* e in

questo modo le genera automaticamente; altri formati, come quello utilizzato, richiedono una gestione manuale.

### 2.2.5 Libreria per la calibrazione della camera

Esistono vari metodi e librerie, free o meno, sviluppate in ambito universitario o aziendale, per il processo di Camera Calibration. Abbiamo identificato un algoritmo semplice ed adatto al nostro processo, free ed integrabile poichè scritto in linguaggio *C*: *View Correlation* [BOG94]. Questo algoritmo prevede che l'utilizzatore catturi 5 o più punti di controllo in una foto ed i corrispettivi nell'ambiente 3D; la proiezione dei punti 3D tramite i parametri vista genera dei punti 2D che saranno comparati con quelli della foto. L'algoritmo poi minimizza la distanza tra le coppie di punti corrispondenti modificando i parametri della camera 3D e di conseguenza la matrice di proiezione, procedendo iterativamente finchè l'errore medio non risulta sotto una determinata soglia. L'output di questo processo saranno dei parametri camera che modificheranno questa in modo che risulti identica, con una certa tolleranza, alla fotocamera reale che avrà scattato la fotografia digitale utilizzata.

### 2.2.6 Environment Mapping

Punto nodale della tesi, come spiegato nei capitoli precedenti.
L'Environment Mapping è una tecnica di *"illuminazione basata su immagini"* ampiamente utilizzata per mappare su oggetti riflettenti della scena l'ambiente circostante. Quando questo non è creato tramite modellazione 3D ( caso in cui le riflessioni su oggetti solitamente vengono calcolate mediante ray tracing; oggi però in questo caso si utilizza anche l'environment mapping, effettuando a run-time 6 scatti ortogonali dal centro dell'oggetto ), si utilizzano delle foto che, mappate sulle sei facce di un cubo, rappresentano a 360° l'ambiente che circonda la mesh; ogni punto di questa prenderà il colore dato dal texel corrispondente nel cubo secondo un vettore riflessione calcolato. Questa tecnica

è però un'approssimazione del reale poichè si basa su due presupposti non sempre veri e che possono falsare i risultati:

1. Tutte le radianze degli oggetti dell'ambiente che incidono sull'oggetto centrale provengono da una distanza infinita. Quando questo non si verifica, la riflessione di oggetti vicini può essere non corretta. Inoltre non si presenta il fenomento di parallasse, per cui oggetti vicini sembrano spostarsi rispetto ad oggetti lontani al movimento della camera.

2. L'oggetto centrale è convesso in modo che non ci siano auto-riflessioni. Al contrario queste auto-riflessioni non compaiono.

Il focus del progetto è però sulla cattura della componente diffusa della radianza dell'ambiente per ottenere una illuminazione base dei mobili ( raramente riflettenti ) e di considerare quindi il texel colpito dalla normale delle facce della mesh e non quello dato dal vettore riflessione, come descritto nella libreria usata per il Tone Mapping. La bontà ed i relativi problemi di questo tipo di illuminazione saranno poi discussi in seguito.

### 2.2.7 Filtro di convoluzione e libreria per la gestione di immagini

Dovendo estrapolare il contributo diffuso dalle fotografie usate per il cube mapping in modo da generare una "mappa di luce" o *Diffuse Cube Map*, dove ogni vertice delle mesh prende il valore del texel colpito dalla sua normale, si è pensato di procedere testando il risultato dell'applicazione di un filtro di convoluzione ( con dimensione molto grande ) a questi scatti, ipotizzando che il risultato potesse fornire una buona approssimazione di una reale mappa di luce diffusa ( non era possibile utilizzare strumenti esterni per ottenere risultati certi; inoltre abbiamo preferito non includere eventuali librerie, come quella rilasciata da ATI, troppo corpose e complesse ). La convoluzione, infatti, miscela per ogni texel il contributo dei suoi vicini, secondo determinati pesi scelti a priori; in questa maniera si costruisce una mappa di luce dove

ogni punto ha la sua illuminazione originale più le componenti di luce delle zone vicine. Questo metodo può simulare le varie luci che incidono in ogni punto di un oggetto ( un modello reale vorrebbe che ogni punto sia illuminato dal totale dei texel presenti sulla semisfera centrata nella normale uscente da quel punto, come descritto in [NVDIA10] ).
Il metodo non è esatto ma fornisce generalmente dei risultati veritieri; varie applicazioni impiegano metodi di convoluzione simili ma con differenti filtri, ottenendo approssimazioni al reale più o meno buone. Come già accennato, un'applicazione che crea un diffuse cube map realistico è fornito da ATI; integrare questo nel codice avrebbe però richiesto molto tempo e risultati non necessariamente così differenti dal metodo scelto; si è preferito proseguire perciò con la convoluzione semplice.

Per lavorare facilmente su immagini in formato .jpg è stata aggiunta al progetto la libreria free *OpenCV* che fornisce ottimi strumenti per queste elaborazioni. Inoltre si aveva già una conoscenza di questo tool.

### 2.2.8 Libreria per l'uso del Tone Mapper

Avendo deciso di utilizzare Ogre come motore grafico, la prima scelta per l'uso di un tone mapper che potesse migliorare la resa di oggetti sintetici sovrapposti ad una fotografia in modo da fornire un risultato il più realistico possibile, è ricaduta sulla libreria *Ogre-HDRlib* [LUK07]. Questa libreria sfrutta l'Ogre Compositor ( framework che permette di definire facilmente effetti full screen post-procedurali ) per creare effetti HDR ( come già spiegato in precedenza l'High Dynamic Range è una tecnica utilizzata in grafica computerizzata e in fotografia per consentire che i calcoli di illuminazione possano essere fatti in uno spazio più ampio, un high range appunto, e si possano rappresentare valori di illuminazione molto alti o molto bassi, in modo da migliorare contrasti e colori; presenta però dei contro, tra cui l'effetto "cartoon" che spesso assumono le immagini ). In particolare, di questa libreria, tra le varie opzioni disponibili, sono stati sfruttati il Tone Mapper per la modifica

della colorazione dei vari componenti della scena e il Key per la modifica della luminosità media della scena.
Questa libreria fa un pesante uso di script in linguaggio HLSL per creare questi effetti post-procedurali.

### 2.2.9 DirectX e HLSL

Le *DirectX* sono procedure che formano un set di comandi, funzioni e strumenti specifici per la grafica 3D e l'utilizzo di schede grafiche. Vengono sfruttate tramite linguaggi di programmazione per shader, cioè che operano direttamente all'interno della pipeline grafica, modificando quello che è il risultato finale dell'ambiente 3D con effetti di diversi tipi, tra cui ombre, luci, colori, ecc...; il linguaggio *HLSL - High Level Shading Language*, sviluppato da Microsoft, è stato utilizzato sia nei file richiamati dai materiali degli oggetti per la modifica del colore e delle texture delle mesh, sia dalla libreria HDRlib per il tonemapping.

# Capitolo 3

# Interfacce

Le interfacce sono state progettate durante lo sviluppo dell'algoritmo, in modo da fornire un supporto veloce, intuitivo e facilmente utilizzabile; l'estetica è stata considerata il fattore meno importante. La versione finale prevedere tre interfacce, ognuna adibita ad un compito preciso. Saranno illustrate e mostrate di seguito.

## 3.1 Interfaccia principale - View Correlation

L'interfaccia principale, Figura 3.1, racchiude una window sul mondo 3D ed una window per la visualizzazione di fotografie, entrambe affiancate e della stessa dimensione ( sono presenti due slider per scorrere la fotografia nel caso sia più grande del riquadro che ne visualizza solo una parte, senza scalarla per adeguarsi alle sue dimensioni ). Questo scelta è stata fatta per avere un approccio più semplice alla selezione di punti per il processo di View Correlation; inoltre sono presenti due box che visualizzano le coordinate dei punti di controllo selezionati, una per il 2D ed uno per il 3D affiancati, così da avere sulla stessa linea coppie di punti corrispondenti. É stata data la possibilità di eliminare uno, diversi o tutti i punti selezionati in caso di errore. Infine due button prevedono di lanciare il processo di View Correlation e di settare

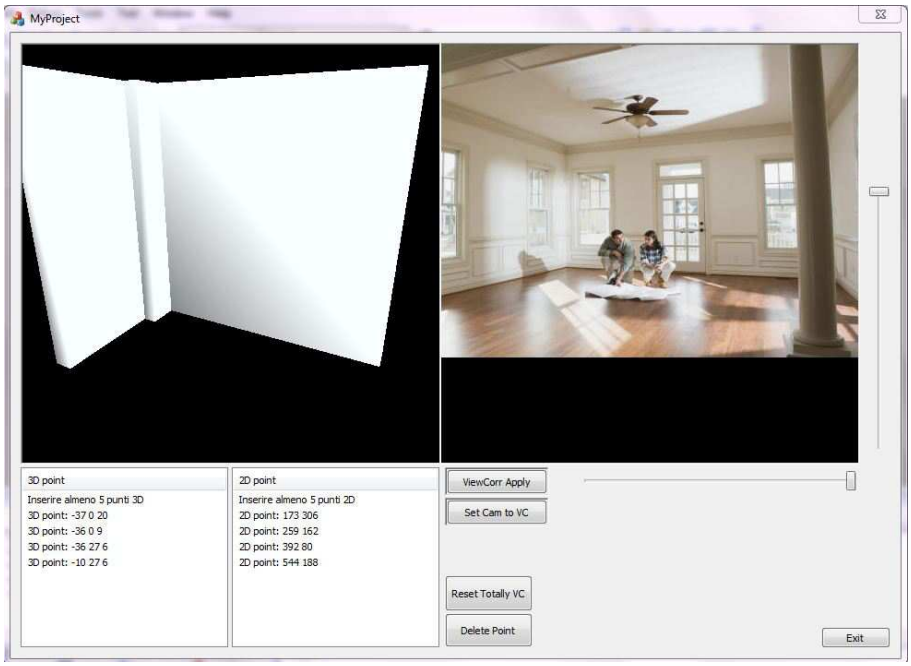


Figura 3.1: *Interfaccia principale - View Correlation*

la camera 3D secondo l'output di questo in una nuova interfaccia che sarà descritta nella sezione "interfaccia render HDR".

## 3.2 Interfaccia generazione Cube Map

Questa interfaccia è stata progettata dopo aver deciso di adottare la modalità di cattura dei 6 scatti aggiuntivi dell'interno fotografato. Lo script di Ogre del materiale genera il cube map a partire o da 6 scatti separati e rinominati corretamente o da un file in formato *DDS*, formato apposito per il cube map; poichè non era possibile creare tale file se non tramite software esterni al progetto, si è dovuto elaborare un modo per gestire gli scatti separatamente. Come si vede in Figura 3.2, sono presenti sei slot per il caricamento dei sei scatti ortogonali; le pareti devono essere inserite nella stessa maniera in cui si trovano nella realtà, come se si guardasse la stanza da sopra ( per facilitare

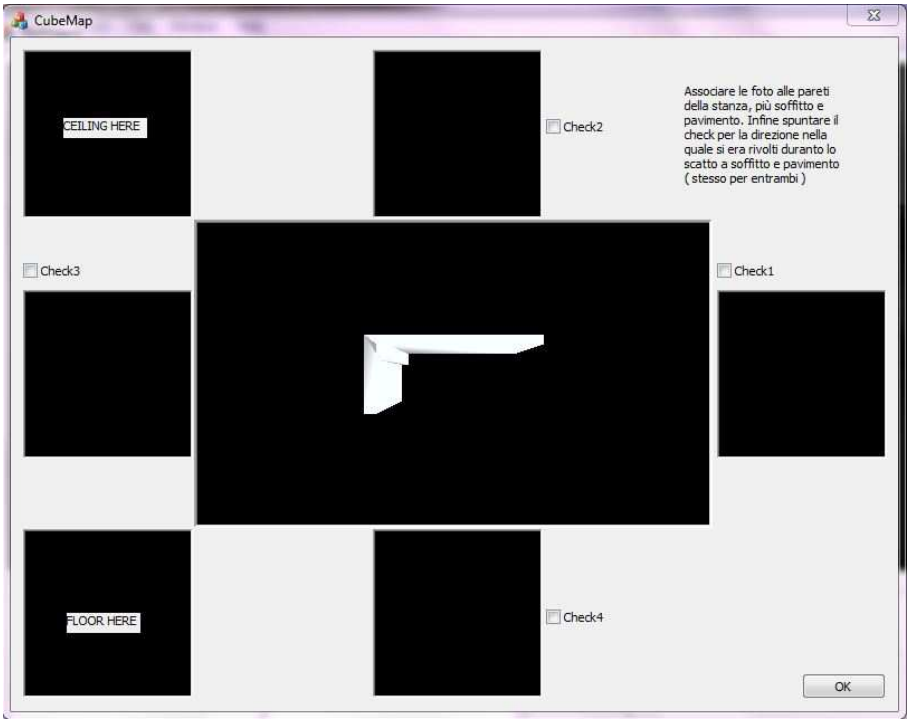


Figura 3.2: *Interfaccia Cube Map*

questo passaggio è presente al centro una vista top della scena 3D contenente le pareti della stanza create come mesh; ogni foto corrisponde così alla propria parete ). Il soffitto e il pavimento hanno il proprio slot indicato. Infine è necessario indicare, tramite checkbox, la parete che si considera come "front", ovvero quella verso cui si era rivolti al momento dello scatto di soffitto e pavimento. Nel prossimo capitolo sarà spiegato in dettaglio come viene creato il cube map da questa interfaccia.

## 3.3 Interfaccia render HDR

Quest'ultima semplice interfaccia, Figura 3.3, presenta una finestra di render secondo i parametri output della View Correlation e una moving window

per la gestione della luminosità media tramite uno slider ed il salvataggio del render. Si è preferito non utilizzare come finestra di render la finestra nell'interfaccia principale poichè il render doveva essere della grandezza della foto stessa e la window della finestra principale non è modificabile in dimensioni.

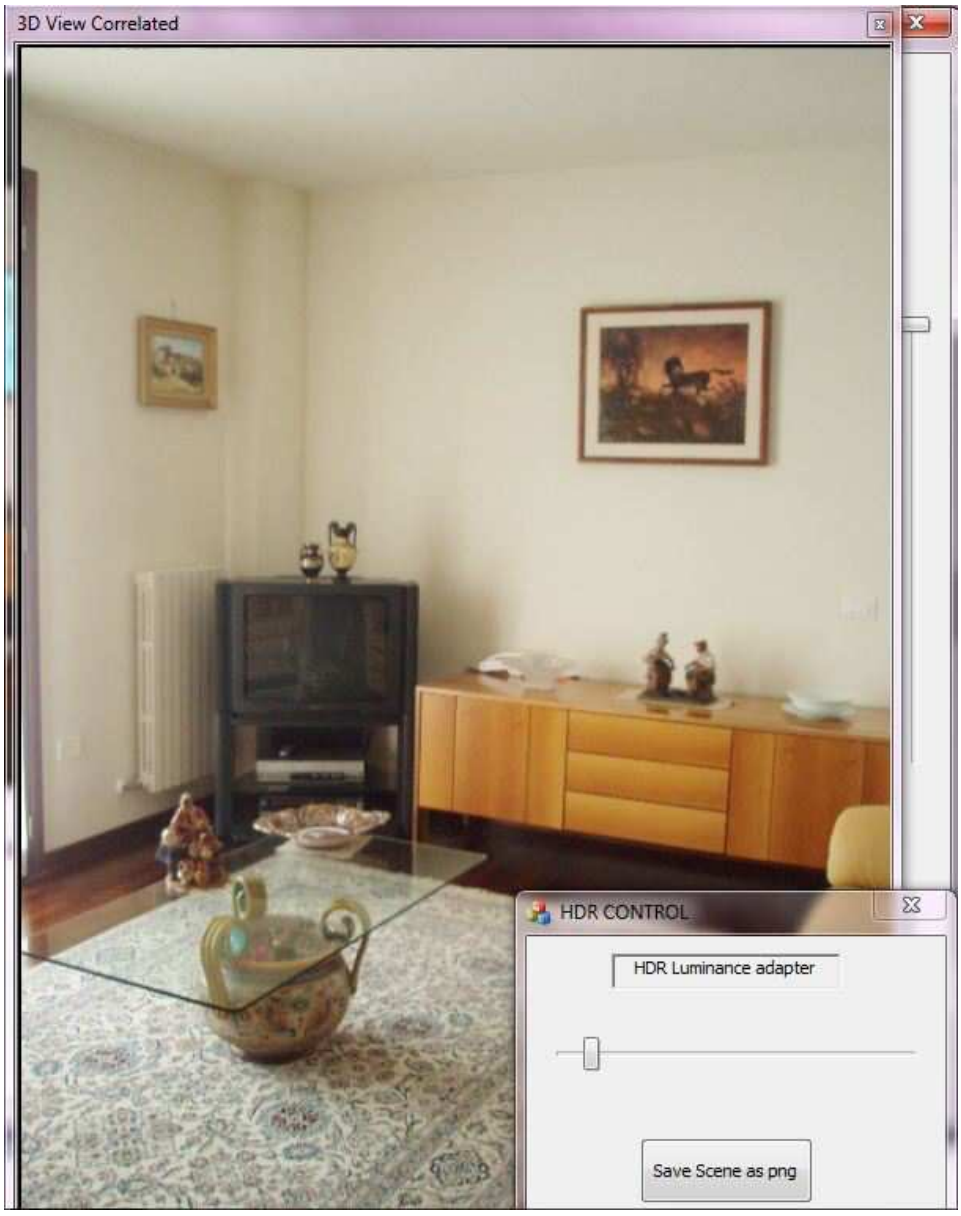


Figura 3.3: *Interfaccia render HDR*

# Capitolo 4

# Sviluppo dell'Algoritmo

Fin'ora si è vista la traccia dell'algoritmo prodotto nelle sue varie componenti e funzionalità, le librerie e le tecniche scelte; infine le interfacce che forniscono un'idea di come si opera per svolgere tutti i passi necessari per ottenere il risultato finale desiderato. Di seguito verrà visto il tutto in maniera dettagliata, illustrando soprattutto quanto definisce l'algoritmo ideato, senza tralasciare il necessario per la realizzazione delle strutture come il motore grafico, le interfacce, ecc..., che verranno visti però in maniera più veloce ( spiegazioni approfondite sono lasciate ai manuali di Ogre e MFC ).

Qui una lista esaustiva dei passi dell'algoritmo che un operatore deve eseguire. La procedura è molto veloce e semplice ed il lavoro è quasi completamente a carico dell'algoritmo; quanto richiesto all'operatore è il necessario per avere uno strumento flessibile e dinamico:

1. INPUT( dal cliente ): foto prospettica di una stanza e relativi 6 scatti ortogonali; misure delle pareti presenti nella foto;

2. Creazione di pareti semplici sul software *Giotto Mobil CAD* ( è sufficiente creare solo quelle visibili nella foto e della lunghezza visibile, non intere ) e inserimento della mobilia scelta dal cliente in posizione corretta;

3. Esportazione separata di pareti e mobilia in formato 3DS;
4. Lancio del software creato;
5. Creazione del Cube Map tramite interfaccia apposita ( caricamento 6 foto più scelta del front );
6. Caricamento della foto prospettica nell'interfaccia base;
7. Scelta dei punti di controllo per il processo di View Correlation ( 5 o più ) nella foto e nel 3D dove sono visualizzate solo le pareti; avvio del processo;
8. Lancio dell'interfaccia di render al termine del processo;
9. Eventuali modifiche di luminosità;
10. OUTPUT: foto stanza con mobili inseriti ed illuminati correttamente.

In Figura 4.1 viene visualizzata la struttura delle classi e delle librerie principali ( quelle inserite direttamente nel progetto come file sorgenti ).

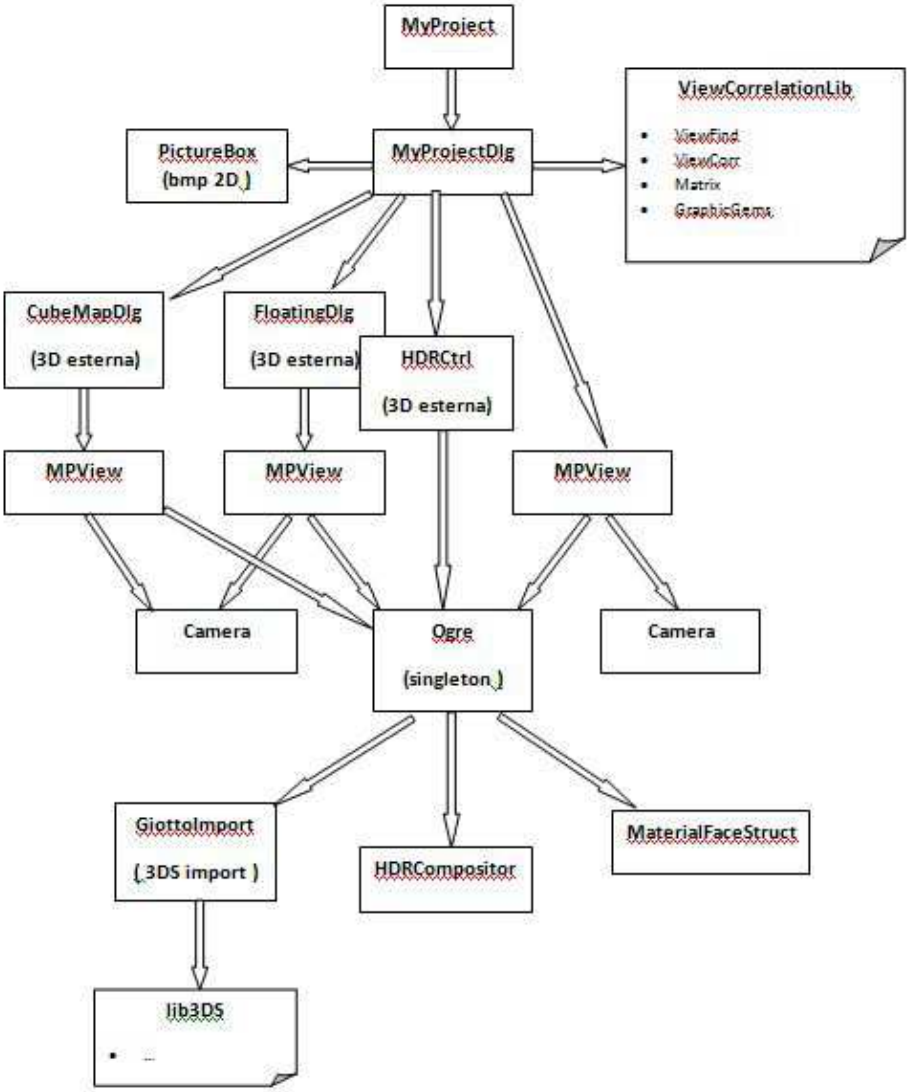


Figura 4.1: *Diagramma Classi*

## 4.1 Substrato generale

L'implementazione dell'algoritmo ha richiesto una progettazione ed una programmazione che fungesse da substrato ad esso; questo è costituito da classi e funzioni di libreria MFC, creazione ed inizializzazione del motore grafico, funzioni per il caricamento di foto digitali.

In particolare evidenziamo, senza soffermarci troppo nel dettaglio, oltre alla classe base MFC che funge da *main()*, una classe MFC per ognuna delle tre interfacce di dialogo, con relativi controlli ed eventi.
Ognuna di esse istanzia ed inizializza la classe relativa al controllo di quanto avviene all'interno della finestra adibita al 3D ( gestione mouse in particolare ) e vi collega il motore di render Ogre ( inizializzato solo alla prima chiamata; la classe è definita secondo il design pattern Singleton ). Ogre viene inizializzato secondo dei parametri base ( citabili sono il *DirectX9 rendering subsystem* e *FFSA - anti aliasing* ), carica dinamicamente le librerie .dll relative ad Ogre, MFC e OpenCV, aggiunge ai path i percorsi utili per il recupero di files; crea inoltre uno *scene manager* per la gestione ed il controllo degli elementi della scena ( come detto Ogre è un motore che fa uso di scene costruite come alberi, dove ogni oggetto rappresenta un nodo ).

Successivamente vengono create delle camere 3D da collegare allo scene manager e alle finestre all'interno delle interfacce; camere che possono essere ruotate, traslate e zoomate e permettono una vista ortogonale o prospettica.

Al caricamento di oggetti 3D in formato 3DS viene effettuata la lettura del relativo file tramite libreria apposita e, sempre sfruttando Ogre, si creano dei buffer contenenti i vertici, le normali, le facce, le mesh, le submesh ed i parametri per il texturing che il sistema di render sfrutta per ricreare la geometria e la topologia della mesh nelle sue varie componenti. Ogni submesh corrisponde a varie parti di un oggetto che condividono lo stesso materiale, andando così a formare le singole componenti dell'oggetto stesso: si pensi ad esempio ad un tavolo ( un'unica mesh ) che può avere le gambe in legno, il pianale in marmo o vetro, parti metalliche, ecc.... I vertici inoltre non sono condivisi tra le varie facce ma ognuno di essi è replicato per ogni faccia definita da esso. Vengono infine creati i bounding box delle mesh.

Vengono anche creati gli assi del sistema mondo in fase di inizializzazio-

ne, nascosti però poi nei render, come le pareti, per non visualizzare oggetti "estranei" alla foto e alla mobilia. La mobilia, al contrario, è creata dopo il processo di View Correlation poichè non necessaria nella fase precedente.

## 4.2 View Correlation

### 4.2.1 Substrato per la VC

Dopo aver creato quanto già elencato, si è dovuto introdurre altro per la View Correlation.
Innanzitutto è stata aggiunta la finestra per la foto nell'interfaccia principale e vi si è collegata una classe per la gestione di questa.
Poi, come si vede dall'interfaccia, i due box per la raccolta dei punti di snap ed i relativi button di gestione.
É stata introdotta una funzione che crea un rettangolo di "background" al contesto 3D, ovvero una superficie sempre visibile e dietro ad ogni altro oggetto ( renderizzato prima degli altri ), di dimensioni pari ai valori output della VC per la window; questo viene texturato con la fotografia di riferimento.
Infine si è dovuto elaborare un modo per catturare questi punti: se nella foto è sufficiente leggere la posizione del mouse all'interno della finestra, nel 3D si è dovuto creare un metodo ad-hoc. Tramite la funzione:

*Ogre::Camera::getCameraToViewportRay( Real screenx, Real screeny )*

si ottiene un "ray" nello spazio mondo a partire dalla camera e passante attraverso il punto di coordinate normalizzate screenx e screeny sulla viewport, corrispondente alla posizione del mouse ( termina in corrispondenza del far plane del frustum ). Vengono poi valutati tutti i vertici delle mesh presenti ( inizialmente, come detto, sono create solo le pareti della stanza ) e viene considerato quello a distanza minore dal raggio, effettuando il calcolo della distanza del punto dalla retta in questa maniera:

$$distanza(\overline{P \quad ray}) = \frac{|(P - puntosulray) \times ray|}{|ray|}$$

In codice:

*Ogre::Vector3 campointdist = pointP - myray.getOrigin()*
*Ogre::Vector3 projectionlenght = VectProduct(campointdist, myray.getDirection())*

Sappiamo che il modulo del vettore che si ottiene dal prodotto vettoriale, diviso per il modulo del secondo vettore ( *mayray.getDirection* in questo caso ) fornisce proprio la distanza punto retta cercata; poichè il fattore di divisione è sempre costante per tutti i confronti ( il ray lanciato dalla camera è costante ), abbiamo omesso questa operazione.
Si valuteranno quindi i valori di *projectionlenght.lenght()* .
Questo punto infine viene trasformato ( attraverso la moltiplicazione per la matrice di vista e quella di proiezione ) in coordinate 2D nella viewport. In questo modo si può valutare la sua distanza dal cursore del mouse; se questa è sotto una certa soglia definita si procede allo snap su quel punto 3D. Lo snap viene evidenziato tramite la creazione di un billboard, ovvero un oggetto piatto quadrato che pone la sua normale e la sua faccia sempre verso la camera. Il click del mouse memorizza il punto in coordinate 3D.

Inoltre la libreria VC è distribuita stand-alone e prevede che tutto sia fornito staticamente tramite files; l'input e l'output sono previsti da e per uno specifico software di ray tracing, "Rayshade". Questo ha richiesto un elaborato controllo tramite dubugger sulle matrici 3D passate in input e sui parametri output, oltre ad alcune modifiche apportate alle strutture della libreria.

### 4.2.2 Descrizione del processo di VC

Questo processo, descritto in [BOG94], richiede almeno 5 punti di controllo che non siano tutti sullo stesso piano, che la camera 3D sia nell'ottante corretto dello spazio 3D e che la direzione di vista sia verso il centro degli oggetti 3D. Di default la camera è posizionata in adeguata; l'orientamento della stanza e della mobilia nello spazio 3D, non sempre uguale, può richiedere delle rotazioni manuali della camera.
Ottenuti questi punti, si effettua un processo iterativo che consiste nel trovare l'errore per i parametri di vista 3D:

1. i punti 3D sono proiettati in una viewport 2D delle dimensioni della fotografia, tramite i parametri di vista della camera ( matrici T di traslazione e R di rotazione ); i passi sono i seguenti:

   - Il punto P in coordinate mondo, $P_w$, viene calcolato in coordinate camera come:

     $$P_w \times T \times R = [a, b, c] = P_{cam}$$

   - Ora $P_{cam}$ viene riportato in coordinate 2D, $P_s(u, v)$, relative alla fotografia, tramite le seguenti equazioni:

     $$u = -\frac{x_s \times a}{c \times \tan\theta} + x_c$$

     dove: $x_s$ è la metà della larghezza dello screen ( foto ); $\theta$ è la metà dell'angolo di vista orizzontale; $x_c$ è il centro dello screen sull'asse X.

     $$v = -\frac{r \times x_s \times b}{c \times \tan\theta} + y_c$$

     dove: $r$ è l'aspect ratio della camera; $x_s$ è la metà della larghezza dello screen ( foto ); $\theta$ è la metà dell'angolo di vista orizzontale; $y_c$ è il centro dello screen sull'asse Y.

- Le equazioni sono poi riscritte ed utilizzate in maniera leggermente differente per ottimizzare la fase successiva; riscrivendo $P_{cam}$ come:

$$P_w \times R - E \times R = [a, b, c] = [x - er_x, y - er_y, z - er_z]$$

dove: $E$ è il punto 3D di vista.

e ponendo:

$$d_s = \frac{1}{\tan\theta}$$

otteniamo:

$$u = -\frac{d_s \times x_s \times (x - er_x)}{z - er_z} + x_c$$

$$v = -\frac{d_s \times r \times x_s \times (y - er_y)}{z - er_z} + y_c$$

2. i punti ottenuti dopo questi passaggi vengono poi comparati con i punti catturati nella fotografia per creare un set di *"error term"*; se l'errore medio è sotto una determinata soglia, l'iterazione termina, altrimenti si procede;

3. sono calcolate le derivate parziali di $u$ e $v$ rispetto ai 10 parametri "iterativi": $er_x$, $er_y$, $er_z$, $\phi_x$, $\phi_y$, $\phi_z$, $d_s$, $r$, $x_c$, $y_c$. I parametri $\phi$ sono rotazioni individuali attorno agli assi e, pur non essendo esplicitati nelle equazioni precedenti, sono racchiusi in $x$, $y$, $z$. Queste derivate riempiono una matrice Jacobiana ( matrice che ha per termini le derivate parziali di una funzione ); si invertire poi questa matrice e la si moltiplica per il set di *"error term"* per trovare i nuovi valori di correzione. Questi valori di correzione vengono infine sottratti dai correnti parametri di iterazione per creare un nuovo set di parametri di vista;

4. si usano questi parametri per settare la camera 3D; il processo poi ricomincia, finchè l'errore medio non è sotto una determinata soglia o in due iterazioni successive questo non cambia.

Il processo prevede che in output, oltre i parametri camera, venga creato uno screen grande contenente la scena, ma che venga visualizzata solo una window più piccola della dimensione dell'immagine originale. Abbiamo simulato questo in Ogre ( che differisce da Rayshade e non gestisce tutti questi valori direttamente ) creando una Viewport della grandezza dello screen, visualizzata però dentro una window di un'interfaccia MFC della dimensione della foto originale e traslata opportunamente per centrare perfettamente la scena.

Di seguito viene presentato il file contenente i parametri di input ed output di un esempio di processo di VC effettuato:

**INPUT:**

eye point -18.355146 20.918242 46.754349
look point -18.621265 20.709780 45.813217
up vect -0.056722 0.978031 -0.200597
d/s ratio 1.732051 – aspect ratio 1.000000
half screen size 280.000000 373.000000 – xcenter ycenter 280.000000 373.000000
fov 60.000000
num of data points 6
3D point -37.633682 0.000000 20.907850 – 2D point 14.000000 225.000000
3D point -37.633682 27.000000 20.907850 – 2D point 9.000000 709.000000
3D point -36.033680 0.000000 9.707850 – 2D point 152.000000 275.000000
3D point -36.033680 27.000000 9.707850 – 2D point 152.000000 666.000000
3D point -10.433681 0.000000 6.407850 – 2D point 529.000000 215.000000
3D point -10.433681 27.000000 6.407850 – 2D point 555.000000 686.000000

**OUTPUT:**

eye point -6.463604 16.748463 49.478004
look point -6.903443 16.656601 48.584641

up vect -0.064891 0.995406 -0.070407
d/s ratio 2.460520 – aspect ratio 1.029589
screen size 562.000000 886.000000
window size 2.000000 0.000000 561.000000 744.232422
fov 44.378109 63.972233

### 4.2.3 Visualizzazione dei risultati del processo di VC

Vogliamo presentare infine delle semplici immagini che fanno capire quale sia l'output della View Correlation. In Figura 4.2 si vede un cubo di Rubik usato come foto per testare la VC; è stato quindi creato un cubo 3D, visualizzato in modalità wireframe ed applicata la VC. Le Figure 4.3 e 4.4 mostrano il risultato di due test. Si nota come il processo, pur non essendo preciso al millimetro, fornisce una buona calibrazione della camera 3D.

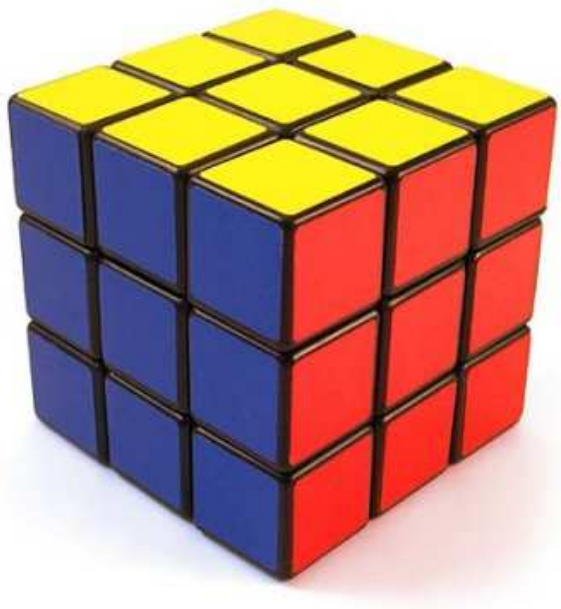

Figura 4.2: *un cubo di Rubik*

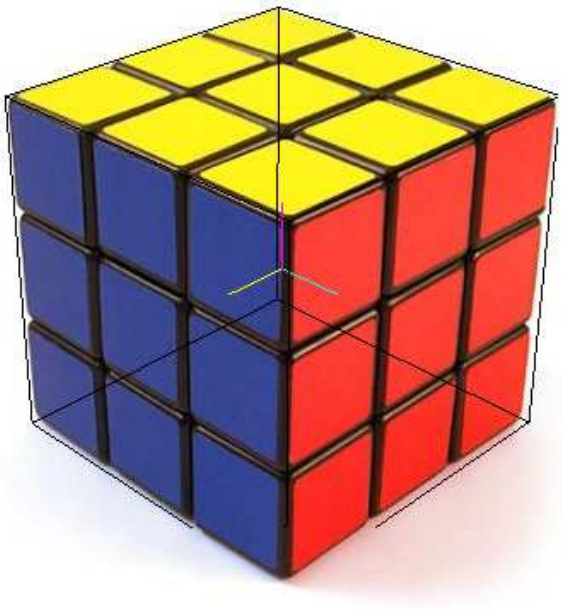

Figura 4.3: *cubo di Rubik e cubo 3D wireframe dopo VC - 1° test*

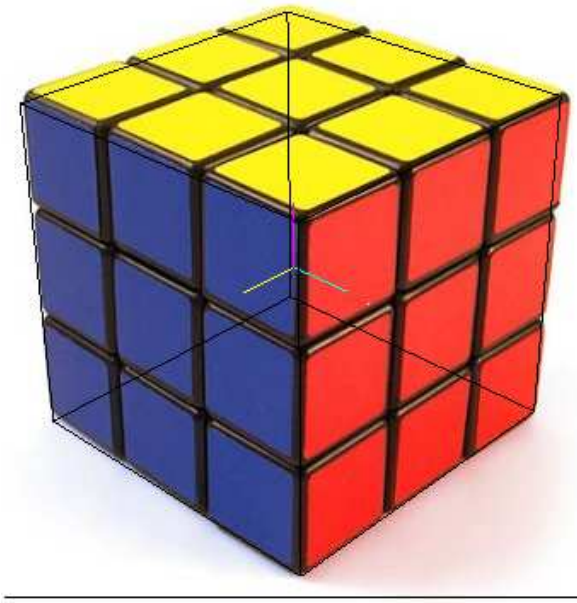

Figura 4.4: *cubo di Rubik e cubo 3D wireframe dopo VC - 2° test*

## 4.3 Diffuse Cube Map

### 4.3.1 Substrato per il Cube Map

É stata creata l'interfaccia, tramite una classe apposita, come descritto nel capitolo relativo, completa di finestre per il caricamento delle immagini e della finestra 3D con vista top sulle pareti. La classe è relativamente semplice e non richiede spiegazioni più dettagliate sulle strutture o sulla programmazione. Tramite la classe Ogre, utilizzando la libreria *OpenCV*, si procede alle operazioni che descrivono l'algoritmo ( rotazione, flip, ecc... ) e a quella di smooth dell'immagine per ottenere la componente diffusa.

### 4.3.2 Descrizione del processo di creazione del DCM

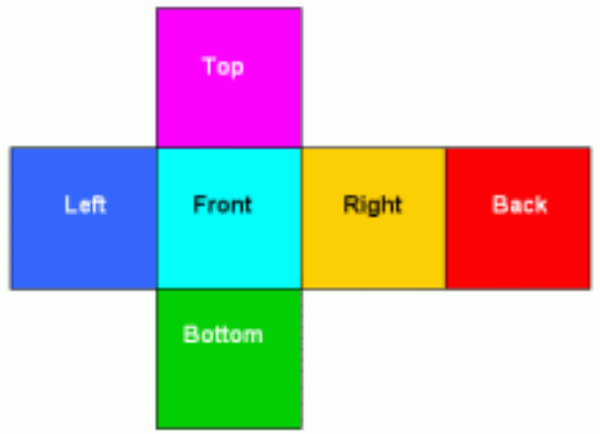


Figura 4.5: *struttura cube map*

Il modo in cui si caricano le foto nell'iterfaccia è stato pensato per risultare facile ed intuitivo, soprattutto per chi non conosce il cube map, la cui struttura è mostrata in Figura 4.5; infatti la posizione del front non è predefinita ( e così anche le altre ), ma può variare a seconda di come sono stati effettuati gli scatti e di come è orientata la mesh delle pareti in vista top. Tale struttura perciò non corrisponde direttamente a quella del cube map, come inteso in

grafica. É necessaria quindi una “traduzione”; l’algoritmo elaborato si divide nei seguenti passi:

1. Tutte le foto vanno specchiate poichè le texture del cube map saranno mappate sugli oggetti della scena in modo da generare l’effetto riflessione ( specchio );

2. Dopo aver identificato lo scatto *front*, i due scatti *left* e *right* andranno invertiti di posizione;

3. Occorre ruotare l’ *up* ( soffitto ) e il *down* ( pavimento ) secondo questa logica: la parte bassa del soffitto corrisponde alla foto *front*, così come la parte alta del pavimento. Quindi:

   - Se nell’interfaccia lo scatto *front* è nello slot in basso allora non sono necessarie rotazioni;
   - Se lo scatto *front* è nello slot alto, allora pavimento e soffitto vanno rotati di 180°;
   - Se lo scatto *front* è nello slot di destra, allora il pavimento viene ruotato di 90° in senso orario, il soffitto in senso antiorario;
   - Se lo scatto *front* è nello slot di sinistra, allora il pavimento viene ruotato di 90° in senso antiorario, il soffitto in senso orario;

   Questo garantisce di avere sempre una riflessione esatta sugli oggetti.

4. Infine si effettua lo smooth delle foto per simularne la componente diffusa; questo si ottiene tramite un filtro gaussiano ( che interpola i valori in maniera non lineare, ma secondo una funzione gaussiana appunto, dove il texel centrale porta il contributo maggiore e allontanandosi da esso il contributo via via diminuisce per i texel più lontani ) di ampiezza pari alla dimensione minima dell’immagine. Questo il codice:

*int gaussianfiltersize = min(src→width, src→height);*
*cvSmooth(src, src, CVGAUSSIAN, gaussianfiltersize, ...);*

5. Le immagini così ottenute vengono infine salvate secondo la modalità richiesta da Ogre ( ricordiamo che la richiesta di cube map prevede o il formato DDS, file unico, oppure le sei singole componenti con stesso nome più l'aggiunta della posizione nel cube map, ovvero $\_fr$, $\_bk$, $\_lf$, $\_rt$, $\_up$, $\_dn$ ). Di seguito il codice che crea il cube map all'interno di uno script per un materiale:

```
texture_unit
{
      cubic_texture   cube_map.jpg   combinedUVW
                ⋮
}
```

### 4.3.3 Visualizzazione dei risultati del Cube Map

Verranno visualizzate di seguito delle immagini rappresentative del processo di creazione del *Diffuse Cube Map* e della sua applicazione su una sfera 3D.

Figura 4.6: *fotografie di un ufficio, posizionate come nel reale*

Figura 4.7: *cube map dell'ufficio ricreato per il texturing di oggetti*

Figura 4.8: *diffuse cube map dell'ufficio ricreato per il texturing di oggetti*

Figura 4.9: *sfera a cui è applicata la texture*

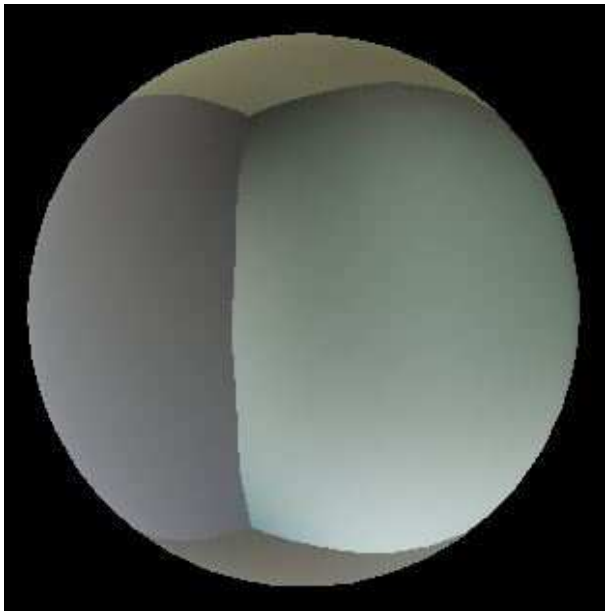

Figura 4.10: *sfera a cui è applicata la texture diffusa*

## 4.4 Render HDR

### 4.4.1 Substrato per il render HDR

Oltre l'interfaccia di render HDR, comprensiva di finestra 3D e finestra di gestione della luminosità media, il lavoro principale svolto è stato quello di inclusione nel progetto della libreria apposita e delle relative inizializzazioni; inoltre sono stati creati i files di configurazione per l'aggiunta dei path in Ogre ed aggiunte tutte le directory utilizzate da questa libreria:

*/Directory_di_progetto*
*/media*
***/material***
***program***
***script***
***texture***

*/models*
*/photo*
***/HDRCompositorScripts***
***program***
***script***

Ogni cartella è indicativa del suo contenuto ( models per le mesh 3D; photo per le foto del cliente; material per i materiali delle mesh scritti su files; HDRCompositor per gli script che gestiscono il Tone Mapping ).
Tutta la procedura segue il processo di View Correlation; infatti, dopo aver ottenuto i parametri output da questa, si eseguono i seguenti passi:

1. Viene creata l'interfaccia di render delle dimensioni adatte;
2. Vengono poste ad invisibili le pareti 3D;
3. Viene settata la camera secondo i parametri della VC;
4. Viene creata la struttura di background con la fotografia;
5. Si avvia ed inizializza la libreria HDR;
6. Infine vengono create le mesh dei mobili.

### 4.4.2 Descrizione del processo di elaborazione del render HDR

Per realizzare effetti schermo post-procedurali, Ogre fornisce un framework chiamato *Compositor*. Una volta istanziato il Compositor per una viewport, è possibile creare dei file script che definiscono gli effetti che il Compositor applicherà. Ogni script conterrà un certo numero di *technique* ( come quelle dei materiali ) che descrivono diversi approcci per creare l'effetto cercato ( a seconda delle capacità della scheda grafica ); ognuna di esse presenta diversi

passi, ovvero tutte le operazioni necessarie per generare la resa; queste vengono applicate una alla volta fino a comporre il risultato finale.

Ribadiamo che il contributo di questa libreria vuole essere quello di creare una scena, mista di elementi sintetici e immagini reali, con dei colori adeguati e simili tra i vari oggetti, per eliminare la differenza percettiva tra le due categorie. Questo avviene cercando di imitare il comportamento fisico della luce, calcolando i colori e la luminosità in HDR; invece di una profondità di colore a 8 bit, tutti i calcoli sono effettuati in valori floating point. Il "dynamic range" è il rapporto tra il massimo ed il minimo valore. Nel contesto naturale questo valore è elevato e per visualizzarlo su un normale display viene utilizzata una scala logaritmica che mappa questi valori da high a low dynamic range ( o color space ). L'intero processo è computazionalmente costoso e necessita di dispositivi grafici con shader 2.0 e supporto per render target in valori floating point.

I passi eseguiti dalla libreria sono i seguenti:

1. La scena è renderizzata su una superficie floating-point;
2. Viene effettuato un downsample della scena di un fattore 4, per ulteriori step;
3. Viene calcolata la media logaritmica della luminosità della scena dal downsample;
4. La scena finale viene "disegnata" con il Tone Mapper selezionato che trasforma l'HDR nel colorspace del dispositivo.

La progettazione e programmazione di una libreria che utilizza pesantemente questi script HLSL e il Compositor di Ogre richiede grande conoscenza della materia, come anche la conoscenza della percezione umana. Per questo saranno illustrate di seguito solo le tecniche utilizzate in questo progetto, ovvero il Tone Mapping e la gestione e modifica della luminosità media. Ulteriori

approfondimenti sono lasciati a [LUK07], dove sono illustrate altre tecniche per ottenere ulteriori effetti visivi che simulano il reale comportamento di superfici.

- La luminosità media della scena viene calcolata mediante la seguente formula:

$$\overline{L}_w = \exp(\frac{1}{N}\sum(\log(\delta + L_w(x,y))))$$

dove $L_w(x,y)$ è la media pesata della luminosità di un pixel tramite il modello YUV ( color space ) che simula meglio la percezione umana del colore rispetto al modello RGB.
Ora conoscendo la luminosità media della scena è possibile mappare lo spettro HDR in uno LDR; questo lo si può ottenere tramite un mapping lineare di questo tipo:

$$L_{scaled} = key \times \frac{L_w}{\overline{L}_w}$$

dove il valore *key* è un valore chiave che permette di "aggiustare" la luminosità dell'immagine finale.

- Vari Tone Mapper sono implementati in questa libreria, ognuno con i suoi punti di forza e di debolezza; la scelta su quale utilizzare è stata effettuata "ad occhio", preferendo quello che ha reso, alla vista, nel modo migliore una scena di riferimento usata per questo test. L'*Adaptive Logarithmic Mapping* usa un mapping che segue la relazione:

$$L_d = \frac{\log_x(L_w + 1)}{\log_x(L_{max} + 1)}$$

In questo modo tutti i valori sono mappati in un range di $[0, 1]$; questa compressione dei valori, se troppo elevata, tende però a far perdere un po' di contrasto ( base x = 2 fornisce un buon contrasto, base x = 10

fornisce una buona compressione ).
Il colore finale si ottiene come:

$$Color = \frac{key}{\log_{10}(\overline{L}_w + 1)} \times \frac{log(L_w + 1)}{log(2 + ((\frac{L_w}{\overline{L}_w})^{\frac{log(0.7)}{log(0.5)}}) \times 8)}$$

Questo Tone Mapper, rispetto agli altri, fornisce un buon livello di contrasto e mantiene ottimamente tutti i dettagli della scena. I colori però sono poco saturati e l'immagine risulta tendente al grigio. In generale non c'è una soluzione ottima o esatta.

La creazione di materiali tramite script segue lo stesso processo e modalità degli script del Tone Mapper. Per questo progetto è stato creato un materiale di default tramite script; questo viene caricato tramite Ogre durante la creazione delle mesh, clonato per ogni tipo di materiale letto dal file 3DS e infine modificato secondo i vari parametri di questi. Di seguito lo script che crea questo materiale di default e la parte principale di quello che esegue le operazione dello shader:

```
material defaultHDRmaterial
{
  technique
  {
    pass
    {
      vertex_program_ref VSconvenv

      fragment_program_ref PSconvenv

      texture_unit //DIFFUSE CUBE MAP
      {
        cubic_texture stanza.jpg combinedUVW
        tex_address_mode clamp
```

```
            colour_op   modulate
            tex_coord_set   1
        }
        texture_unit //BASIC 2D TEXTURE
        {
            texture   null.jpg
            colour_op   modulate
            tex_coord_set   0
        }
    }
  }
}


fragment   PSconvenv
{
            ⋮
  float3   reflVect   =   −reflect(vettoreVista, normaleVertice);
      //lookup del texel ottenuto dal vettore riflesso per il contributo speculare
  specColor   =   texCUBE(SpecSampler, reflVect).xyz ∗ SpecIntensity;
      //2D texture - legge i parametri UV
  2dColor   =   tex2D(2dSampler, i.TexCoord.xy).xyz ∗ 2dIntensity;
      //lookup del texel ottenuto dalla normale per il contributo diffuso
  diffColor   =   texCUBE(DiffSampler, normaleVertice).xyz
    ∗DiffIntensity;
      //valore finale
  float3   result   =   SurfColor ∗ (specColor + 2dColor + diffColor);
            ⋮
}
```

Il colore finale dei pixel ( questo codice è scritto per il pixel shader ) è

dato dalla somma ( pesata secondo i singoli coefficienti di intensità, lasciati di default a 1 ) delle 3 componenti texture moltiplicata per la colorazione della superficie della mesh nel punto considerato.

Questa "miscela" di texture e colori permette di ottenere per le mesh una illuminazione basata su texture; l'aggiunta del processo di Tone Mapping rende "più veritiera" all'occhio la colorazione totale della scena.

### 4.4.3 Visualizzazione dei risultati del Tone Mapper

Verranno presentate di seguito delle immagini derivanti dal processo. La Figura 4.11 presenta una sfera riflettente inserita in una foto senza il tone-mapper; le seguenti Figure 4.12, 4.13 e 4.14, presentano la stessa sfera ( la posizione è leggermente differente poichè sono stati eseguiti due processi di VC e in entrambi la camera è stata mossa a mano per mostrare maggiormente lo sfondo, altrimenti la sfera avrebbe coperto quasi completamente il background ) con il tone mapper attivo, a vari livelli di luminosità media.
Si nota come la gestione della luminosità cambi radicalmente l'immagine, permettendo di valutare ed apprezzare vari aspetti della scena ( parti più illuminate e parti in ombra ); eventuali scatti di questo tipo che simulano foto ad esposizioni differenti di una macchina fotografica, se combinati assieme, possono crare una foto HDR.
La colorazione, invece, non sembra portare delle modifiche significative; si nota come la Figura 4.13 sia molto prossima alla Figura 4.11, solo leggermente più illuminata e, in questa come nelle altre due, con colori meno saturati e vividi, tendenti al grigio.

Figura 4.11: *sfera su uno sfondo senza tone mapper*

Figura 4.12: *sfera su uno sfondo con tone mapper adaptative log, fattore luminosità basso*

Figura 4.13: *sfera su uno sfondo con tone mapper adaptative log, fattore luminosità medio*

Figura 4.14: *sfera su uno sfondo con tone mapper adaptative log, fattore luminosità alto*

# Capitolo 5

# Risultati

In questo capitolo saranno presentati i risultati di questo progetto; si vedranno dei render ottenuti per varie stanze, evidenziando i relativi problemi riscontrati. In particolare ci sono dei problemi concettuali nella gestione del Diffuse Cube Map, problemi non intuiti in fase iniziale ma identificati solo fase di render.
É inoltre importante sottolineare che tutti i passaggi fatti ( dal caricamento in Ogre, al lavoro per il Cube Map, al salvataggio finale e durante la trasformazione in formato .eps ) tendono a diminuire la qualità dei render finali e della foto di background soprattutto. Alcune di esse poi non hanno una qualità ottimale poichè non sono state scattate con un'alta risoluzione.

## 5.1 Presentazione render finali

In Figura 5.1 sono state inserite delle sfere sintetiche in mezzo a quelle reali della foto; queste sfere hanno una colorazione marrone chiaro con applicata una texture "legno". L'applicazione del Diffuse Cube Map su queste prevede di illuminarle secondo le luci di questa stanza. Essendo le sfere molto piccole rispetto alla scena ne è stata creata una più grande per una migliore visualizzazione.

Figura 5.1: *render biliardo*

Notiamo come il processo di camera calibration sia ottimo in questa situazione, fornendo un posizionamento delle sfere tale da far sembrare queste veramente appoggiate sul tavolo.
Non si può dire altrettanto dell'illuminazione: sebbene le sfere, non riflettenti, presentino, come evidente dalla sfera più grande, luci ed ombre corrispondenti alle relative zone della stanza ( e questo era lo scopo finale ) e il "mix" delle varie componenti del colore e delle texture sia buona, lo "stacco" tra le varie texture del Cube Map è troppo evidente per garantire un effetto "veritiero" ( questo è il primo dei problemi legati al Cube Map che però, come vedremo, può essere evitato ).
Inoltre l'assenza di ombre delle sfere sul piano evidenzia maggiormente gli oggetti sintetici.
Infine si nota abbastanza la mancanza di un processo di anti-aliasing; sebbene questo sia stato attivato inizialmente ( FSAA ), si è reso necessario disattivarlo poichè crea conflitto ( genera un run-time error ) con gli effetti post procedurali tramite shader. Questo perchè l'FSAA di Ogre lavora renderizzando ad una risoluzione e con un livello di downsampling maggiori rispetto al

Compositor. L'unico modo di farli lavorare assieme sarebbe creare uno script che tramite il Compositor e lo Shader ricrei l'effetto di anti-aliasing.

In Figura 5.2 e Figura 5.3 si vede lo stesso render, a due differenti luminosità, di un soggiorno con degli oggetti sintetici ( una clessidra, una scacchiera ed un cono ) appoggiati su un tavolino ed un quadro, anch'esso sintetico, appeso al muro tale da coprire un quadro reale presente nella foto.

Figura 5.2: *render soggiorno*

Figura 5.3: *render soggiorno*

Queste immagini mostrano nuovamente come i processi di creazione, di camera calibration e di "merge" delle varie componenti che forniscono la colorazione funzionino ottimamente.
Purtroppo si notano alcuni problemi: la foto a luminosità maggiore, pur non essendo "sovraesposta", per usare il termine fotografico, ma anzi più prossima alla vera luce della stanza, annulla i contributi del Diffuse Cube Map sugli oggetti piccoli, saturandone il valore bianco. Nell'altra foto, invece, si notano le differenti zone di illuminazione, soprattutto sulla clessidra. In entrambe, il

quadro prende una inaspettata colorazione azzurra, forse a causa di un lavoro errato del Tone Mapper.
In questi scatti si evidenzia il secondo grosso problema del Diffuse Cube Map: come detto, il lookup nel Cube Map avviene tramite la normale di ogni vertice; ma in questo modo oggetti piatti avranno tutti i vertici con la stessa normale e di conseguenza verranno tutti texturati con lo stesso texel, generando così delle mesh con facce monocromatiche, quando invece oggetti tondi, con normali differenti ad ogni punto, avranno un corretto mapping dell'ambiente circostante. Per ovviare a questo problema, riscontrato dopo i primi render di questo tipo, si è cercato di modificare il lookup andando a "distorcere" leggermente queste normali, tramite il vettore di riflessione, in modo da ottenere texel differenti per ogni vertice. Gli oggetti "piatti" in questo modo non risultano più monocromatici ma, come si vede bene dal quadro, il texturing non è esatto. Probabilmente con accorte modifiche di distorsione è possibile ottenere un effetto più realistico, anche se è necessaria una tecnica differente per un corretto lookup.

Figura 5.4: *render ufficio a bassa luminosità*

In Figura 5.4, Figura 5.5 e Figura 5.6 portiamo un altro esempio di risultato finale del processo, a varie luminosità. I due divani ed il tavolino sono posizionati al centro della stanza non paralleli alle pareti, in modo da evidenziare maggiormente le riflessioni dell'environment mapping. Si nota come la mobilia sia inserita correttamente all'interno della stanza vuota; queste mesh sembrano infatti sfiorare le pareti senza andare oltre ed essere appoggiati al pavimento ( anche se guardando con attenzione il pavimento sotto il divano di sinistra si nota come questo sia illuminato dalla luce della vetrata quando dovrebbe invece presentare l'ombra del divano ). Inoltre si può osservare come la luminosità della mobilia si adatta correttamente a quella della stanza tramite la funzione apposita dell'*HDRlib*. Questo integra in maniera ottima gli oggetti al contesto fotografico.

Figura 5.5: *render ufficio a media luminosità*

Non siamo soddisfatti però del comportamento del Cube Map per lo stesso motivo dell'immagine precedente: i divani ed il tavolino, per la maggior parte piatti, mostrano l'errato comportamento del lookup. Con le normali non alterate, d'altro canto, si sarebbero visti degli oggetti monocromatici e

privi di profondità. Il Tone Mapper inoltre non fornisce una colorazione così ottima da far confondere gli oggetti sintetici con quelli reali; essi presentano infatti una colorazione ad effetto "cartoon", non realistica. Si era già detto che questo può essere uno dei problemi dell'HDR.

Figura 5.6: *render ufficio a medio-alta luminosità*

Mostriamo infine un ultimo test: lo scatto è stato effettuato in una cucina e riprende il tavolo e le due pareti "disadorne". Sono state create e posizionate diverse mesh: un tv lcd appeso al muro, degli oggetti appoggiati sul tavolo, un mobile nella parete di fondo affianco alla porta. La Figura 5.7 e la Figura 5.8 presentano il render ottenuto con il metodo del Diffuse Cube Map, a bassa luminosità il primo, a media luminosità il secondo. Si nota come gli oggetti siano inseriti correttamente; ben visibile sull'anta del mobiletto nella prima immagine il problema del Diffuse Cube Map con le normali distorte. Si nota però, tramite questo, come le singole parti del Cube Map rispecchiano l'ambiente circostante: la faccia "back" di questo, visualizzata al centro dell'anta, è più chiara ed illuminata delle altre poichè ritrae le finestre. L'illuminazione

della stanza, però, è troppo bassa; migliorando questo parametro ed ottenendo un valore più simile a quello reale durante lo scatto delle foto ritroviamo il problema della sovraesposizione degli oggetti, cosa che satura i colori al bianco.

Figura 5.7: *render cucina a bassa luminosità*

Figura 5.8: *render cucina a medio-alta luminosità*

In Figura 5.9 è stato effettuato il render senza il Cube Map: le mesh assumono un colore più realistico ma ovviamente non riportano un'illuminazione coerente con l'ambiente ( quella che si nota sugli oggetti è fornita in automatico da Ogre, pur senza l'aggiunta di luci nella scena, per garantire un effetto di profondità ). C'è un problema sulla texture 2D sullo schermo tv, probabilmente a causa di un non corretto mapping UV.

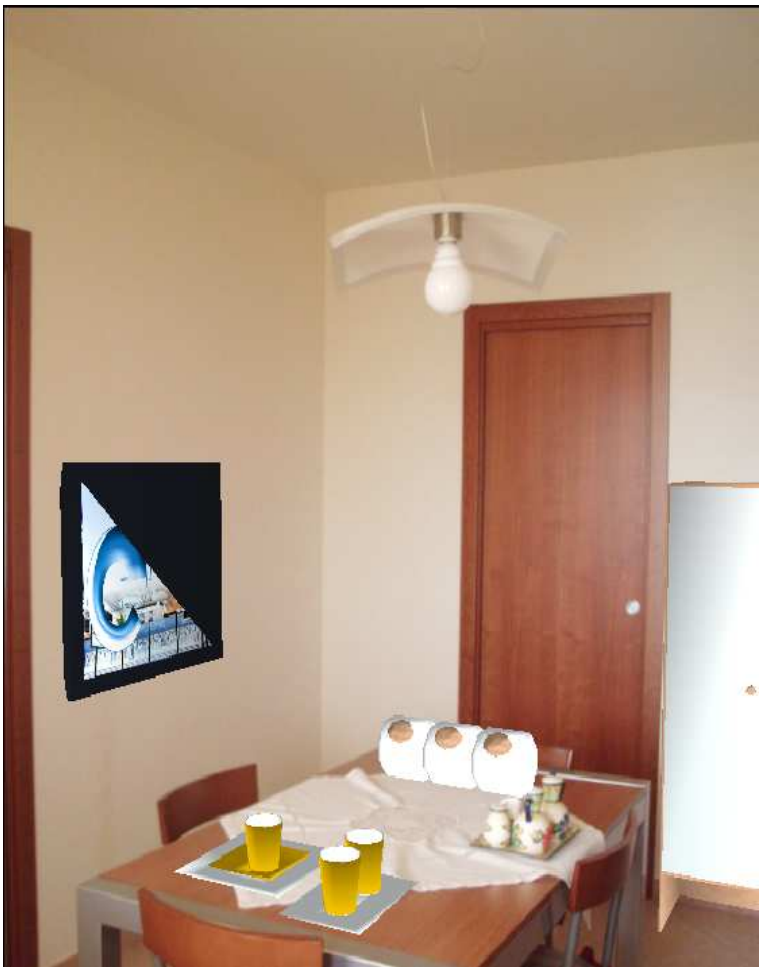

Figura 5.9: *render cucina senza Cube Map*

## 5.2 Valutazioni delle tecniche usate e problemi riscontrati

Sono già stati spiegati i pregi delle tecnologie utilizzate ed i vincoli iniziali; non andrò a ripetere perciò queste cose. É invece opportuno elencare i pregi delle scelte effettuate e del lavoro di programmazione. Quanto creato e prodotto, seppur migliorabile e rivedibile per una presentazione commerciale, fornisce un buon e veloce processo per un render finale, assieme ad un'interfaccia con una buona usabilità. Il software è stato sottoposto ad un testing

da parte di un paio di persone esterne al progetto, fornendo loro il materiale iniziale necessario ed una breve spiegazione delle funzionalità. Il risultato è stato più che ottimo, senza che si incappasse in problemi o si richiedessero ulteriori spiegazioni.
Tutti i processi che fanno parte di questo progetto, una volta inseriti correttamente ( cosa non sempre facile ed immediata ), hanno fornito quanto necessario per testare la resa finale dell'algoritmo.
I problemi principali sono dovuti, come già detto, al Diffuse Cube Map: infatti, letteratura a riguardo esplicita come non si possa ottenere un risultato adeguato per superfici piane da una semplice mappa come quella usata; inoltre la struttura di partenza non sono fotografie catturate come fatto in questo progetto ma sempre un Light Probe. Questo fa sì che il progetto non sarebbe potuto riuscire seguendo appieno le richieste iniziali. Quanto ottenibile è un primo approccio utilizzabile per revisioni e migliorie future per la parte del texturing.
In particolare per creare un Diffuse Cube Map che funzioni correttamente è necessario sfruttare tecniche molto avanzate, quali *Irradiance Map* e *Armoniche Sferiche*, come in [RAHA01], [DEB03], [TAT05], [NVDIA10]. Queste, oltre migliorare il risultato finale, permetterebbero anche, a nostro avviso, di continuare il lavoro su un Cube Map creato come nel nostro caso, eliminando la soglia netta visibile sulle mesh tra le varie facce del cubo e probabilmente anche la necessità di estrapolare con un filtro di convoluzione la componente diffusa delle fotografie.
Inoltre riteniamo opportuno, per migliorare ulteriormente il processo e la correttezza delle riflessioni, effettuare una traslazione durante il look-up del texel del Cube Map: infatti vertici in posizioni diverse ma con stessa normale otterranno sempre lo stesso valore ( e questo può andare bene per un Cube Map di un ambiente esterno molto vasto, come il cielo, come avviene nella demo dell'*HDRlib* ), mentre riteniamo che la loro posizione nel mondo ( soprattutto in interni dove le posizioni degli oggetti incidono maggiormente ) debba riflettersi sul look-up. Nel caso di interni infatti, poichè il Cube Map creato è

assimilabile alla stanza nelle sue dimensioni, è sicuramente possibile trovare una proporzione tra la posizione di un vertice nel mondo 3D ed all'interno del Cube Map ( dove è sempre centrale ), ottenibile appunto effettuando una traslazione del look-up e recuperando un texel differente.
Le foto, che non si avvicinano a quanto si può ottenere da un Light Probe in termini di qualità, hanno il vantaggio di essere semplici e veloci da scattare poichè sfruttate da una modalità tollerante, che non richiede la precisione assoluta. Sono infatti sufficienti a ricreare una illuminazione prossima a quella reale e combinando queste con le tecniche avanzate appena descritte si ritiene possibile raggiungere un livello buono di resa. Il contro di queste è l'impossibilità di fotografare per intero l'ambiente, in particolare soffitto e pavimento; così facendo se si volesse creare l'effetto riflessione su un oggetto si noterebbe la mancanza di continuità tra due scatti vicini.
É inoltre assente l'effetto ombra/occlusione tra oggetti differenti e tra varie parti di un oggetto concavo; questo perchè non c'è interazione tra facce delle mesh durante il Cube Mapping ( come avverrebbe invece tramite un render con ray-tracing ). L'aggiunta di questo porterebbe una maggior consistenza al render finale. Esistono algoritmi e librerie che simulano la ridotta illuminazione di oggetti a causa dell'occlusione ambientale ed alcune tecniche che ricreano le ombre di oggetti 3D da "inserire" nella scena. Ci si è documentati su questi ma non è stato possibile introdurli nel progetto.
Infine si potrebbe includere la riflessione di oggetti sintetici su altri oggetti: una tecnica che si utilizza in grafica consiste nella resa, a partire dal centro dell'oggetto in questione, di 6 immagini ortogonali per ricreare un cube map della scena 3D da aggiungere al cube map ambientale. Questo processo è però molto dispendioso poichè richiede appunto sei rese per ogni oggetto riflettente e viene utilizzato sono in alcuni ambiti, come nella resa precalcolata di alcune scene per videogiochi.

# Capitolo 6

# Sommario

## 6.1 Review

É stato presentato un lavoro di tesi svolto presso un'azienda di sviluppo e gestione di un software di grafica 3D per arredamento di interni. É stato creato un plugin per ampliare le funzionalità del software proprietario, tramite l'utilizzo di strumenti free e integrabili tra loro.
Il software creato prevede, tramite delle interfacce supportate dalla libreria MFC, la possibilità di inserire oggetti sintetici all'interno di una fotografia di interni, come nel caso di mobili che vanno ad arredare un appartamento vuoto; inoltre, si è cercato di ricreare degli effetti di illuminazione su questi oggetti.
Dopo l'acquisizione dei dati input ( fotografia da "arredare", fotografie ortogonali alla stanza scattate dal punto di inserimento dei mobili, misure stanza ), nel software proprietario viene velocemente ricreata una mesh corrispondente alle pareti visualizzabili nella foto primaria; vengono poi inseriti i mobili secondo le richieste. Quindi il tutto è esportato in formato 3DS.
Importando questi file nel software ricreato, si procede applicando il processo di camera calibration ( tramite la libreria View Correlation ) in modo da avere una vista nel 3D identica a quella della fotografia. Si nascondono le pareti 3D e si crea un background con la fotografia. Così facendo i mobili sembrano

realmente posizionati nella fotografia.
Si procede poi con l'illuminazione: escluso l'utilizzo di un classico ray-tracing, si procede con un'illuminazione basata sull'ambiente. Viene generato un Diffuse Cube Map dalle sei fotografie ortogonali modificate secondo un filtro di convoluzione; questo è poi utilizzato per texturare le mesh presenti nella scena, ricreando così l'effetto luci presenti nella stanza. Il contributo luce è inoltre miscelato con il colore e texture originali delle mesh.
L'utilizzo di un Tone Mapper vuole, infine, modificare la colorazione dell'intera scena ( mesh e foto ) in modo da ricreare delle tonalità di colore più simili tra loro ( la differenza tra i colori della foto e delle mesh è sempre molto evidente ).
Come detto, si è deciso di non utilizzare la classica struttura di dati input per il Cube Map, ovvero il Light Probe; inoltre la diffusione ed il look-up dei texel nel Cube Map sono stati ricreati in maniera intutitiva.
I vantaggi di queste modalità scelte sono la velocità e la facilità di utilizzo ( richiesta fatta dall'azienda per avere un prodotto finale usabile e commerciabile ); i contro riguardano soprattutto la fase di illuminazione, non esatta e con dei problemi non facilmente risolvibili se non tramite strutture e algoritmi più avanzati.

Questa procedura ha richiesto un training ed un'ampia ed attenta fase di programmazione sugli strumenti di base: MFC per le interfacce, Ogre per il motore grafico, libreria View Correlation, libreria HDRlib, shader e programmazione HLSL, libreria OpenCV.
Sono state prodotte circa 4300 LOC ( linee di codice ), senza contare librerie e file aggiuntivi.

## 6.2 Sviluppi futuri

Sviluppi futuri riguardano sicuramente la parte finale del progetto, essendo tutto il resto soddisfacente. Sarebbe opportuno modificare, come già spiegato

nel capitolo precedente, il metodo di look-up dei texel nel Cube Map a secondo della posizione dei vertici nello spazio mondo. Inoltre si potrebbe cambiare il Diffuse Cube Map generando una Irradiance Map ed ottenere così una migliore e più precisa resa.
Altri miglioramenti meno importanti potrebbero riguardare l'inserimento di un metodo di Ambient Occlusion in modo da simulare le zone d'ombra che si creano tra gli oggetti a seguito dell'occlusione delle luci. Inoltre si potrebbero creare delle ombre per le mesh in modo da fornire più spessore all'immagine finale; senza ombre infatti non si coglie bene la posizione degli oggetti e si capisce che la prospettiva può ingannare l'occhio umano.

## 6.3 Conclusioni

Pur non essendo soddisfatti del passo finale dell'illuminazione creata a causa dei problemi di look-up, possiamo ritenere che l'approssimazione dell'ambiente tramite foto semplici, così come la creazione del Diffuse Cube Map forniscano uno strumento sufficiente a questa procedura. Purtroppo non è stato possibile implementare nuove modalità di look-up per il Cube Map dopo aver ottenuto i risultati presentati, modifiche che avrebbero reso migliore la resa finale.
Tutte le strutture create ed il codice scritto, invece, funzionano correttamente.
Il progetto ha comunque comportato una grossa mole di studio e progettazione; questo ha comportato un notevole know-how all'azienda come al sottoscritto. Conoscenze che si spera di poter utilizzare ed ampliare in futuro.

# Bibliografia

[BOG94] Rod.G.Bogart. "View Correlation" in: *Graphic Gems II*, James Arvo (ed.), Academic Press, 1991. pp. 181-190.

[GREG96] Gene.S.Greger. *"The Irradiance Volume"*, Cornell University, 1996.

[DEB98] Paul.E.Debevec. *"Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography"*, SIGGRAPH 1998.

[RAHA01] Ravi.Ramamoorthi. e Pat.Hanrahan. *"An Efficient Representation for Irradiance Environment Maps"*, Stanford University, 2001.

[CTHD01] Jonathan.Cohen., Chris.Tchou., Tim.Hawkins. e Paul.Debevec. *"Real-time High Dynamic Range Texture Mapping"*, Eurographics Rendering Workshop 2001, London, June 2001.

[BRE02] Chris.Brennan. "Diffuse Cube Mapping" in: *Direct3D ShaderX: Vertex and Pixel Shader Tips and Tricks*, Wolfgang Engel (ed.), Wordware Publishing, 2002, pp. 287-289.

[DEB03] Paul.E.Debevec. *"HDRI and Image-Based Lighting"*, SIGGRAPH 2003.

[TAT05] Natalya.Tatarchuk. *"Irradiance Volumes for Games"*, Game Developers Conference Europe 2005.

[CASH05] Xiaochun.Cao. e Mubarak.Shah. *"Creating Realistic Shadows of Composited Objects"*, Computer Vision Lab - University of Central Florida, 2005.

[LUK07] Christian.Luksch. *"Realtime HDR Rendering"*, TU Wien, 2007.

[NVDIA03] NVIDIA.Corporation. *"The CG Tutorial"*, 2003, http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter07.html, 31 gennaio 2012.

[NVDIA18] NVIDIA.Corporation. *"GPU Gems"*, 2004, http://http.developer.nvidia.com/GPUGems/gpugems_ch18.html, 31 gennaio 2012.

[NVDIA19] NVIDIA.Corporation. *"GPU Gems"*, 2004, http://http.developer.nvidia.com/GPUGems/gpugems_ch19.html, 31 gennaio 2012.

[NVDIA10] NVIDIA.Corporation. *"GPU Gems 2"*, 2005, http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter10.html, 31 gennaio 2012.

[wiki1] Wikipedia. *"Camera Resectioning"*, http://en.wikipedia.org/wiki/Camera_resectioning, 31 gennaio 2012.

[OGRE] Ogre forum. http://www.ogre3d.org/forums/, 31 gennaio 2012.

# Ringraziamenti

Vorrei ringraziare:

il professor Giulio Casciola per il sostegno e la supervisione di questo progetto;

l'ing. Giorgio Andreani, presidente della Computer Office s.r.l., per avermi ospitato nella sua azienda e fornito la possibilità di lavorare a questa tesi;

l'ing. Giovanni Panzieri per avermi seguito durante tutto il progetto come correlatore;

il tutor aziendale Federico Ercolessi per avermi aiutato enormemente ed insegnato;

tutto il resto dello staff aziendale.