ALMA MATER STUDIORUM - UNIVERSITÁ DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI SCIENZE

CORSO DI LAUREA DI SCIENZE
E TECNOLOGIE INFORMATICHE

REALIZZAZIONE DI UN APPLICAZIONE DI PROJECTION MAPPING CON OPENFRAMEWORKS

Relazione finale in
METODI NUMERICI PER LA GRAFICA

Relatore
Prof.ssa DAMIANA LAZZARO

Presentata da
BRUNO DONATI

II SESSIONE

ANNO ACCADEMICO 2013-2014

INDICE

INTRODUZIONE

Il Projection Mapping è una tecnologia che permette di proiettare delle immagini sulla superficie di uno o più oggetti, anche di forma irregolare, trasformandoli in display interattivi. Il suo utilizzo, abbinato a suoni e musiche, permette di creare una narrazione audio-visuale.

La suggestione e l'emozione scaturite dalla visione di una performance di Projection Mapping su di un monumento pubblico, hanno stimolato la mia curiosità e mi hanno spinto a cercare di capire se era possibile realizzare autonomamente qualcosa di analogo.

Obiettivo di questa tesi è perciò spiegare cos'è il Projection Mapping e fornire una serie di indicazioni utili per realizzarne un'applicazione interattiva con OpenFrameworks (un framework open-source in C++) e hardware a basso costo (un computer, un videoproiettore e un sensore Kinect).

Nel capitolo 1 vedremo quali sono le caratteristiche del Projection Mapping inteso come particolare tipologia di realtà aumentata, alcune dei suoi campi di applicazione più diffusi e quali strumenti servono alla sua realizzazione.

Nel capitolo 2 esporremo la teoria e gli algoritmi su cui si basa il Projection Mapping e il meccanismo di calibrazione del sistema.

Verificata la fattibilità da un punto di vista teorico, passeremo nel capitolo 3 all'analisi di un caso reale: un'installazione interattiva realizzata in collaborazione con il collettivo di architettura "La Prima Stanza" di Montiano. Ne valuteremo gli obbiettivi prefissati e motiveremo le scelte dell'hardware e del software utilizzato.

Nel capitolo 4 andremo ad analizzare le caratteristiche di OpenFrameworks, vedremo come creare un applicativo base e come estenderlo, approfondiremo le specifiche del sensore Kinect.

I passi di implementazione, dalla modellazione dell'oggetto da mappare alla sua importazione in OpenFrameworks, uniti a dettagli di implementazione legati alla natura del progetto, verranno esposti nel capitolo 5.

Il capitolo 6 sarà dedicato all'analisi di alcune classi dell'applicazione implementate per gestire le scene, l'interfaccia grafica e i dati multimediali da riprodurre.

Daremo infine nel capitolo 7 indicazioni e suggerimenti su come calibrare il sistema tramite il software realizzato. Vedremo come utilizzare l'applicazione e l'installazione interattiva.

Capitolo 1
PROJECTION MAPPING E REALTÀ AUMENTATA

Per poter comprendere cos'è il Projection Mapping è necessario prima orientarsi fra le varie declinazioni di realtà virtuale esistenti.

Seguendo la tassonomia introdotta da Milgram e Kishino, poniamo perciò l'ambiente reale in cui viviamo e un ambiente completamente virtuale ai due estremi di un intervallo che chiameremo "Continuum della Virtualità". L'insieme continuo dei punti compresi fra i due estremi è definito "Realtà Mista", dove ogni punto ha una componente reale e una componente virtuale. La componente virtuale della realtà mista è costituita da uno strato aggiuntivo di informazioni generato da un computer o da un dispositivo mobile, che permette di arricchire ed aumentare la normale esperienza sensoriale.

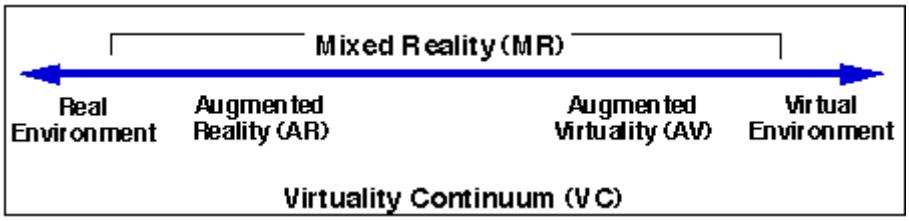


FIGURA 1.1

La Realtà Aumentata è un tipo di realtà mista dove la componente reale è preponderante rispetto a quella virtuale.Il Projection Mapping (conosciuto anche come Spatial Augmented Reality o Video Mapping) è un caso particolare di Realtà Aumentata, dove la componente virtuale anziché essere fruita tramite visori o dispositivi mobili, viene proiettata direttamente su uno o più oggetti reali. Per far ciò è necessario generare un modello tridimensionale dell'oggetto da aumentare e poi gestire il processo di mappatura dell'oggetto virtuale su quello reale. La proiezione permette di modificare alcune

caratteristiche visive dell'oggetto, alterandone la percezione del colore, del materiale e della luce.

1.1 **Campi di applicazione**

Il Projection Mapping permette di trasformare qualunque oggetto in un potenziale schermo di proiezione, sia su piccola che su larga scala. Dato l'elevato livello di impatto visivo, gli utilizzi più comuni sono in campo artistico/architettonico, didattico e pubblicitario.

CHIESA DI SAINT CLIMENT DE TAÜLL – LLEIDA – SPAGNA

PROIEZIONI SU STATUE DURANTE LA NOTTE DEI MUSEI A ROTTERDAM

PROIEZIONE PER LA PRESENTAZIONE DI UN AUTOMOBILE

1.2 **Strumenti necessari**

L’hardware e il software richiesto variano molto in funzione della dimensione e della complessità dell’oggetto da mappare. Ad esempio, nel caso della proiezione permanente nella chiesa di Saint Climent de Taüll, sono stati utilizzati sei proiettori pilotati da un software commerciale proprietario, mentre l’interno della chiesa è stato digitalizzato con un costoso scanner laser 3D.

Nel nostro caso, dovendo lavorare su una proiezione meno complessa e avendo a disposizione un budget decisamente più limitato, è stato sufficiente utilizzare un sistema composto da un pc e un video proiettore. L’oggetto da proiettare è stato modellato manualmente utilizzato un software di modellazione 3D gratuito, così come gratuito è l’ambiente di sviluppo scelto per implementare il software di mappatura.

Vedremo in seguito, più nel dettaglio, le scelte implementative legate al progetto da realizzare.

Capitolo 2
CALIBRAZIONE DEL SISTEMA

Il primo problema da risolvere consiste nel trovare una relazione tra le coordinate del modello virtuale e quelle dell'oggetto reale in modo da poter proiettare efficacemente le informazioni legate all'oggetto virtuale su quello reale.

2.1 **Teoria e Algoritmi**

Il modello che viene utilizzato è chiamato "pin-hole camera" e permette di definire una relazione fra le coordinate tridimensionali di un punto e la sua proiezione su di un piano utilizzando una trasformazione prospettica.

$$sm' = A[R|t]M'$$

In questa trasformazione, M' rappresenta il punto nello spazio tridimensionale, m' il punto proiettato.

A è una matrice di parametri intrinseci dell'ottica, R e t sono rispettivamente una matrice di rotazione e di traslazione dell'ottica rispetto alla scena.

L'unione delle matrici [R|t] è detta matrice dei parametri estrinseci

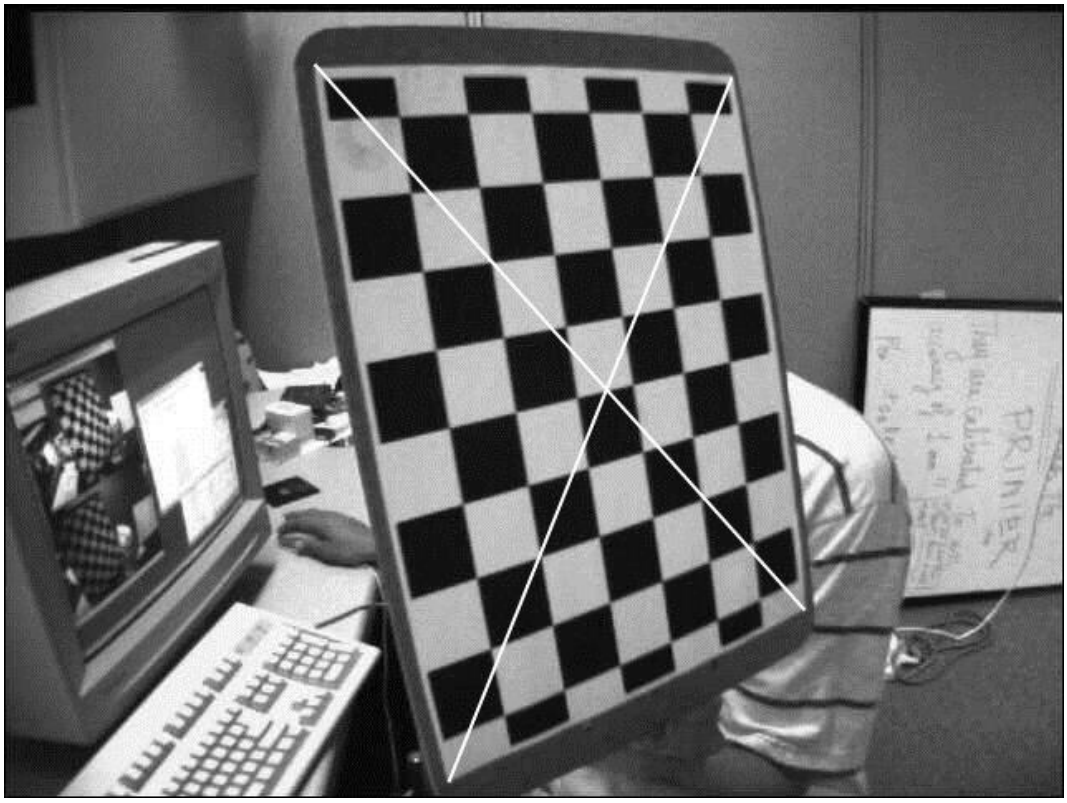
Entrando più nel dettaglio

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

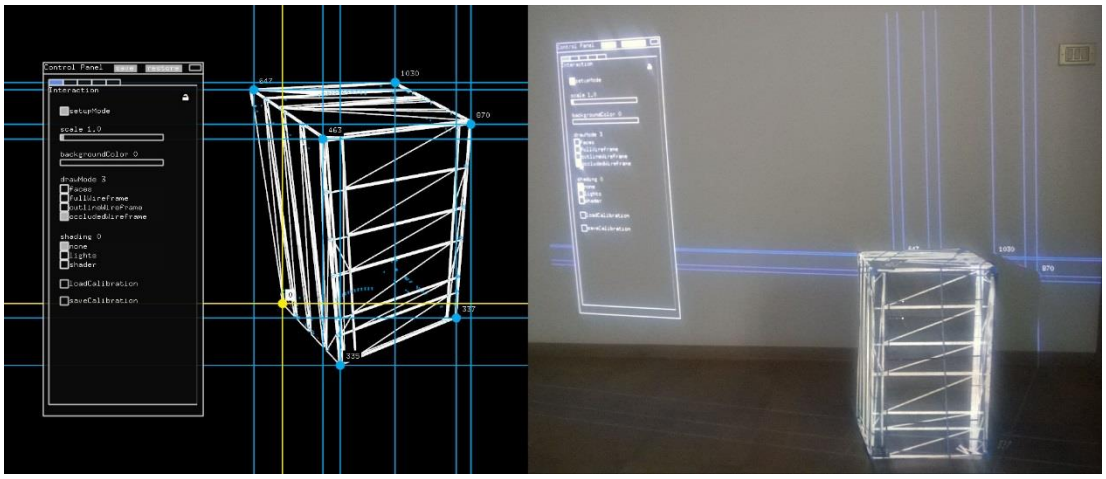
dove X, Y, Z sono le coordinate tridimensionali del punto nello spazio; u e v sono le coordinate in pixel del punto proiettato; f_x e f_y sono le lunghezze focali della lente espresse in pixel; c_x e c_y sono le coordinate del punto principale (di solito il centro dell'immagine); $r_{11}..r_{33}$ sono le componenti della matrice di rotazione; $t_1..t_3$ sono le componenti della matrice di traslazione.

Il modello è fondamentalmente un modello ottico e può essere applicato sia a dispositivi ottici di input (videocamere) che di output (proiettori).

Per poter proiettare quindi il nostro modello virtuale sull'oggetto reale è necessario trovare il valore delle 16 variabili indicate. Ci verrà in aiuto il framework OpenCV tramite una potente funzione di calibrazione che implementa il modello sopra indicato. Andremo però ad utilizzarla con un approccio nuovo: la procedura viene di solito utilizzata per calibrare delle videocamere tramite l'acquisizione di immagini contenenti un pattern noto (chessboard), nel nostro caso invece avremo a disposizione solamente un proiettore, per cui andremo a far collimare in maniera empirica un campione di punti del modello tridimensionale con l'oggetto reale, lasciando poi alla procedura di calibrazione il compito di stimare i parametri intrinseci dell'ottica del proiettore e la sua posizione e rotazione nello spazio.



CALIBRAZIONE DI UNA VIDEOCAMERA TRAMITE PATTERN DI TIPO CHESSBOARD



CALIBRAZIONE EMPIRICA TRAMITE COLLIMAZIONE DI PUNTI

La procedura di calibrazione prende come parametri in ingresso:

- un vettore di vettori tridimensionali contenente le coordinate dei punti scelti per la calibrazione
- un vettore di vettori bidimensionali contenente i valori rilevati manualmente delle corrispondenti coordinate proiettate sul piano
- le dimensioni in pixel dell'immagine proiettata

e restituisce al termine dell'elaborazione:

- la matrice A dei parametri intrinseci dell'ottica
- la matrice R di rotazione
- la matrice t di traslazione

L'algoritmo è suddiviso in tre passaggi

1) Assume un valore iniziale dei parametri intrinseci dell'ottica
2) Stima la posizione iniziale dell'ottica come se i parametri intrinseci dell'ottica fossero corretti
3) Utilizza l'Algoritmo di ottimizzazione di Levenberg-Marquardt per minimizzare l'errore di ri-proiezione, ovvero la somma totale dei quadrati delle distanze fra la proiezione dei punti rilevata manualmente e la proiezione dei punti calcolata utilizzando la stima corrente dei parametri intrinseci dell'ottica.

Restituisce inoltre il valore dell'errore di ri-proiezione.

Capitolo 3
IL PROGETTO DA REALIZZARE

Una volta chiarite le potenzialità della tecnologia e verificata la fattibilità, sono andato alla ricerca di un caso reale da implementare. Dopo un primo positivo incontro con il collettivo di architettura “La Prima Stanza” di Montiano, in cui ho illustrato l’idea di fondo del Projection Mapping, abbiamo iniziato una collaborazione volta a definire l’oggetto e i contenuti da mappare.

3.1 **Il workshop “Vivere il Paesaggio”**

Il collettivo “La Prima Stanza” ha organizzato in collaborazione con la Scuola di Ingegneria e Architettura di Cesena un workshop di progettazione inerente l’architettura del paesaggio.

Diversi gruppi di studenti hanno elaborato 6 progetti di micro-architettura volti a rileggere lo spazio pubblico del parco V. Battistini di Montiano. Una commissione ha poi giudicato i progetti e decretato il vincitore.

UN MOMENTO DEL WORKSHOP “VIVERE IL PAESAGGIO”

3.2 **Un albero molto particolare**

Il progetto vincitore è risultato il progetto “Agri-Cultura” che rappresenta una reinterpretazione concettuale di un albero, realizzato con materiale di recupero e che può essere utilizzato con diverse configurazioni e utilizzi (espositore, contenitore per lo scambio di libri e cd, lampione a basso costo)

IL PROGETTO “AGRI-CULTURA”

L’opera è stata realizzata in diversi esemplari, uno dei quali appositamente dedicato al Projection Mapping, e installata nel parco pubblico di Montiano.

L’OGGETTO DA MAPPARE

Si è poi svolta l’inaugurazione dell’opera durante un evento denominato “Territolio” con degustazioni di olio e prodotti eno-gastronomici di aziende del territorio, dove ha avuto luogo anche la performance di Projection Mapping denominata “Territolio 2.0”.

UN MOMENTO DELL’EVENTO “TERRITOLIO”

3.3 **Obbiettivi del progetto**

Partendo dalle premesse del progetto “Agri-Cultura”, abbiamo deciso di riproporre nell’installazione di Projection Mapping alcune delle idee che ne hanno ispirato la realizzazione. Il nostro oggetto mappato doveva perciò diventare una sorta di chiosco multimediale interattivo, che potesse raccontare al pubblico come è stato concepito e alcune delle sue potenzialità.

Nell’idea dei progettisti, uno degli utilizzi dell’oggetto è di essere utilizzato come un espositore, volto allo scambio di libri o cd. Nell’installazione abbiamo voluto concettualizzare queste idee, trasformando l’installazione in un proiettore di citazione letterarie e in un lettore musicale interattivo.

L’oggetto doveva anche raccontare i passi che hanno portato alla sua realizzazione, perciò è stata inserita anche una sezione dedicate al backstage del workshop e un sezione di crediti con l’elenco dei partecipanti e dei collaboratori al progetto.

Infine, l’opera doveva in qualche modo interagire con il pubblico, permettendo all’utente di scegliere le funzioni della stessa senza necessariamente utilizzare i tradizionali dispositivi di input (mouse e tastiera):

3.4 **Scelte Hardware e Software**

Per la realizzazione del progetto è stato utilizzato hardware di uso comune, sia perché già presente nelle mie disponibilità, sia perché di qualità comunque sufficiente per una buona realizzazione dell’installazione.

Il Pc utilizzato è un DELL con processore Intel Core i5, 6GB di RAM, una scheda video NVIDIA GeForce GT220 con un 1GB di RAM dedicata.

Il sistema operativo è Windows 7 SP1.

Il video proiettore utilizzato è un BenQ W1000+ con luminosità di 2000 ANSI Lumen. Le sue caratteristiche hanno permesso di mappare senza troppi problemi l’intero oggetto e buona parte della parete posta alle sue spalle.

Sono risultati particolarmente utili come accessori, un treppiede con supporto universale orientabile per video proiettore e una tastiera wireless Logitech K400r per il controllo dell’installazione da remoto.

IL SETUP COMPLETO

IL SETUP VISTO DA UN'ALTRA ANGOLAZIONE

Per la parte di interazione con il pubblico, si è scelto come dispositivo di input un sensore Kinect riciclato da una console Microsoft XBox360.

Per rendere maggiormente chiaro agli utenti quale fosse l'area sensibile all'interazione si è deciso di delimitarla con altre cassette della frutta del tutto simili a quelle che costituivano l'opera da proiettare.

UN DETTAGLIO DELL'AREA DI INTERAZIONE

La scelta del software di gestione dell'installazione è stata orientata su di una soluzione multipiattaforma, open source e ad elevate prestazioni. Dopo un'attenta valutazione si è optato per OpenFrameworks, un framework open source in C++ orientato alla programmazione "creativa".

OpenFrameworks permette un approccio semplificato alle librerie di grafica OpenGL e alla libreria di computer vision OpenCV, fornisce classi di gestione per immagini, video, audio, font e file Xml; può inoltre essere esteso da add-on creati da terze parte per compiti più specifici, è disponibile per Windows, MacOsx e Linux, supporta gli IDE XCode, Code Blocks, Eclipse e Visual Studio 2012 su cui è ricaduta la scelta.

La modellazione dell'oggetto da mappare è stata affidata a SketchUp su indicazione del collettivo di architettura "La Prima Stanza", mentre per il ridimensionamento e il fotoritocco veloce delle immagini è stato usato Paint.Net. Si è infine utilizzato Audacity per piccole modifiche ai file audio.

Capitolo 4
OPENFRAMEWORKS

OpenFrameworks è un framework aperto rilasciato come sorgente non compilato e tuttora in fase di sviluppo. La versione corrente è la 0.8.3 ed compatibile con Visual Studio 2012, si consiglia di verificare al momento dello scaricamento e dell'installazione quale sia la versione di Visual Studio consigliata perché altrimenti potrebbero insorgere problemi in fase di compilazione.

Vedremo ora com'e strutturato il framework e come generare una soluzione per sviluppare un applicativo base, ne analizzeremo la struttura e vedremo come estenderne le funzionalità tramite degli add-on.

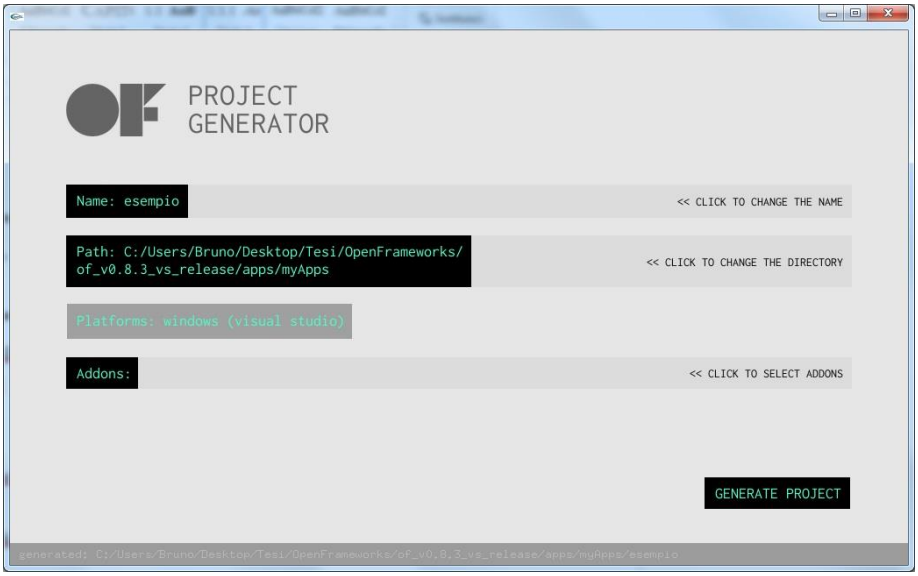
4.1 **Installazione e struttura del framework**

Il processo di installazione del framework è piuttosto triviale: è sufficiente scaricare dall'indirizzo http://www.openframeworks.cc/download/ la versione del framework compatibile con il proprio sistema operativo e ambiente di sviluppo, e decomprimerla in una cartella.

Le cartelle principali del framework sono la cartella "apps" dove vengono collocati i progetti di Visual Studio delle applicazioni da sviluppare, la cartella "examples" che contiene esempi che illustrano le funzionalità del framework, la cartella "addons" in cui vanno collocate le estensioni del framework e la cartella "projectGenerator" che contiene un generatore di progetti che permette di semplificare e velocizzare notevolmente la creazione di un applicativo minimale da utilizzare come base per lo sviluppo dell'applicazione.

4.2 **Creazione e struttura di un'applicazione base**

Il modo migliore per creare un'applicazione base in OpenFrameworks è di utilizzare il project generator fornito con il framework stesso. Si tratta di un applicativo in cui è possibile specificare il nome della soluzione da creare, il percorso di destinazione e gli eventuali addons che si vogliono includere nel progetto. Una volta confermato il comando di generazione, verrà creata una soluzione di Visual Studio che includerà i riferimenti alla librerie di OpenFrameworks e agli eventuali addons selezionati, e i file main.cpp, ofApp.h e ofApp.cpp



L'INTERFACCIA DEL PROJECT GENERATOR

Il file main.cpp costituisce il punto di entrata dell'esecuzione del programma, contiene una chiamata per l'inizializzazione dell'ambiente OpenGL e istanzia la classe principale dell'applicazione ofApp.

```
#include "ofMain.h"
#include "ofApp.h"
//========================================================================
int main( ){
	ofSetupOpenGL(1024,768,OF_WINDOW);			// <--------
setup the GL context

	// this kicks off the running of my app
	// can be OF_WINDOW or OF_FULLSCREEN
	// pass in width and height too:
	ofRunApp(new ofApp());
}
```

Il file ofApp.h contiene le definizioni dei metodi implementati dalla classe ofApp in ofApp.cpp e ci permette già di vedere quali eventi il framework ci mette a disposizione.

```
class ofApp : public ofBaseApp{

	public:
		void setup();
		void update();
		void draw();

		void keyPressed(int key);
		void keyReleased(int key);
		void mouseMoved(int x, int y );
		void mouseDragged(int x, int y, int button);
		void mousePressed(int x, int y, int button);
		void mouseReleased(int x, int y, int button);
		void windowResized(int w, int h);
		void dragEvent(ofDragInfo dragInfo);
		void gotMessage(ofMessage msg);

};
```

Abbiamo a disposizione tre metodi principali che costituiscono l'ossatura dell'applicazione:

- setup() per l'inizializzazione di variabili e classi, viene lanciato una sola volta subito dopo l'istanziazione della classe ofApp
- update() utilizzato principalmente per l'aggiornamento di stati e variabili
- draw() utilizzato per i comandi di disegno su schermo

  Entrambi i metodi vengono lanciati alternativamente in maniera ricorrente dalla classe ofApp.

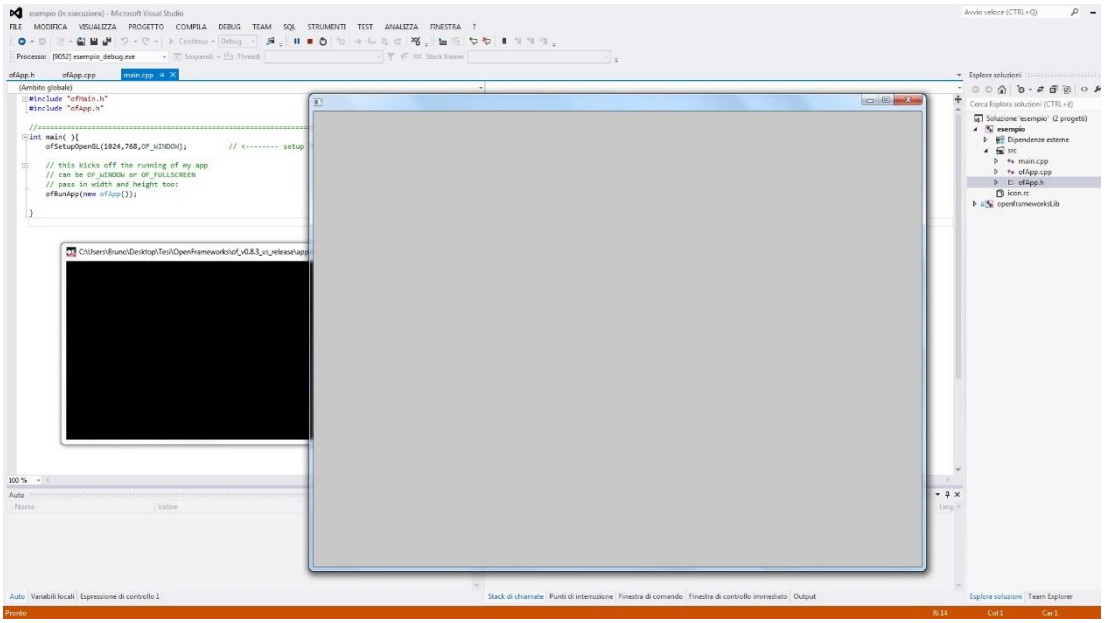
Abbiamo inoltre a disposizione dei metodi per:

- gestire l'input da tastiera (keyPressed(…), keyReleased(…))
- gestire il mouse (mouseMoved(…), mouseDragged(…), mousePressed(…), mouseReleased(…))
- gestire il ridimensionamento della finestra corrente (windowResized(…)),
- gestire il drag and drop (dragEvent(…))
- gestire eventi tramite messaggi (gotMessage(…))

4.2.1 *Prima esecuzione*

Dopo aver aperto la soluzione in Visual Studio possiamo procedere al lancio dell'applicazione. La prima esecuzione richiede un tempo di compilazione superiore, perché il framework viene fornito come sorgente e deve essere interamente compilato, prima di poter permettere l'esecuzione dell'applicazione.

Lanciata l'applicazione otteniamo la visualizzazione di due finestre: una finestra in modalità grafica che contiene l'applicazione vera e propria, e una finestra in modalità console per la visualizzazione di notifiche ed errori di runtime.



LA PRIMA ESECUZIONE DI UN'APPLICAZIONE BASE IN OPENFRAMEWORKS

4.3 **Estendere il framework con gli AddOns (ofx)**

Nonostante il framework sia fornito di un nutrito numero di funzionalità, è uso comune estenderlo con dei componenti aggiuntivi per gli utilizzi più disparati.

Le estensioni sono scaricabili dall'indirizzo http://www.ofxaddons.com, vanno decompresse e collocate nella cartella addons presente all'interno della cartella principale di OpenFrameworks. Generalmente sono fornite di un minimo di documentazione e di qualche esempio di utilizzo.

Ogni add-on riporta la versione del framework per la quale è stato sviluppato, può perciò capitare di dover metter mano ai sorgenti e di scrivere qualche adattamento nel caso si abbia a che fare con add-on sviluppati per versioni precedenti del framework.

Per la realizzazione di questa applicazione di Projection Mapping è stato necessario utilizzare i seguenti addons:

- ofxAssImpModelLoader, per l'importazione di scene e animazioni in formato 3ds, dae, obj
- ofxControlPanel, per una veloce gestione di pannelli di controllo multipli, checkbox, slider e variabili
- ofxCv, per l'interfacciamento alla libreria di computer vision OpenCV
- ofxGui, per la gestione di elementi di interfaccia grafica in OpenGL
- ofxKinectCommonBridge, per l'interfacciamento con le librerie del Kinect for Windows SDK
- ofxProCamToolkit, per l'utilizzo di funzioni di utilità relative alla traduzione di coordinate
- ofxSvg, per la gestione di file in formato grafico vettoriale SVG
- ofxTween, per la gestione delle animazioni
- ofxXmlSettings, per il salvataggio e la lettura di impostazioni in formato XML

4.4 **Microsoft Kinect**

Microsoft Kinect è un sensore che nasce come accessorio della console Microsoft Xbox 360. La sua principale caratteristica è che permette di interagire con la console senza la necessità di impugnare alcun tipo di controller o di indossare dei marker, ma semplicemente utilizzando il movimento del proprio corpo.

KINECT for XBOX 360.

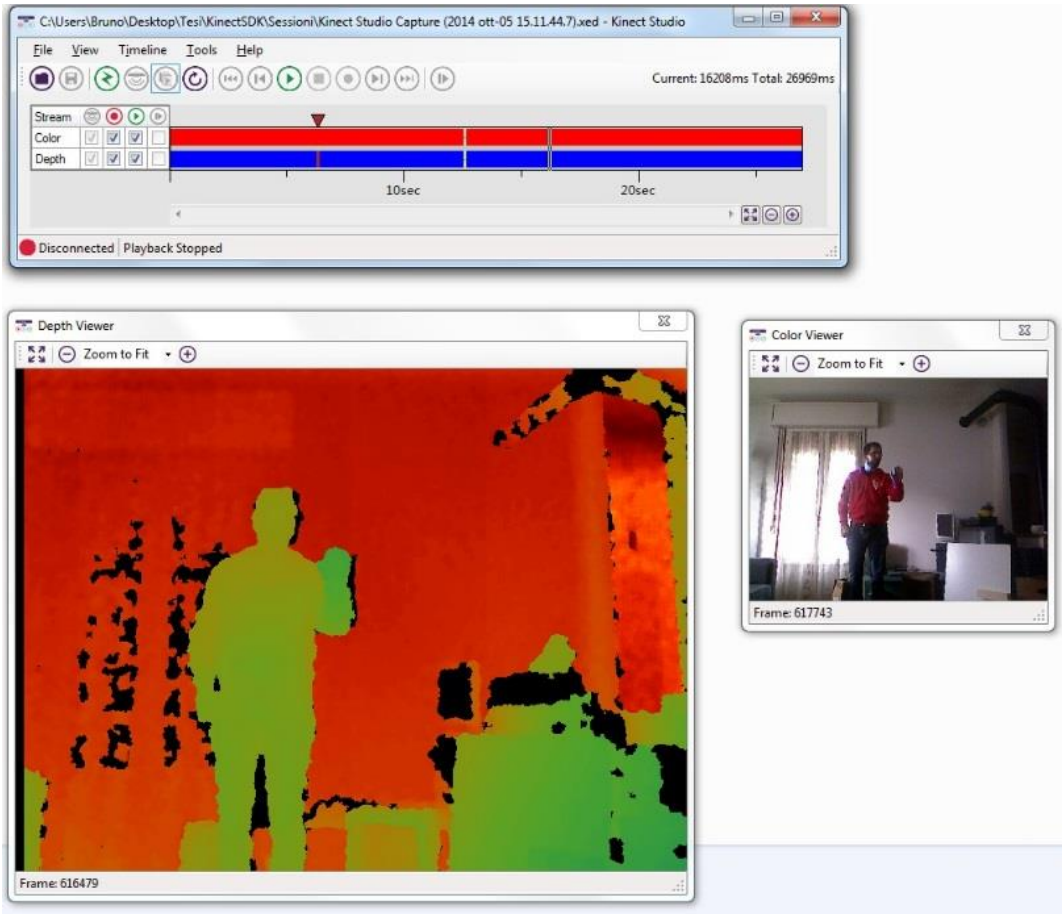
Per rendere possibile tutto ciò, Kinect sfrutta una videocamera RGB, un emettitore e una videocamera a infrarossi, Possiede inoltre un array di microfoni direzionali e una base motorizzata per poter impostare un corretto angolo di acquisizione.

Il flusso di dati video RGB generato dalla webcam e di dati di profondità generati dal sensore a infrarossi, può essere utilizzato in maniera grezza oppure più strutturata: In quest'ultimo caso ci si affida a una libreria di sistema che utilizza i dati grezzi per generare uno scheletro vettoriale di un eventuale soggetto inquadrato dal sensore. E' possibile andare ad analizzare le coordinate dei singoli snodi e utilizzarle per gestire un'interazione con l'utente.

L'ambiente OpenFrameworks supporterebbe nativamente l'interfacciamento con il sensore Kinect di Microsoft, a patto però di utilizzare i driver della libreria libfreenect (http://openkinect.org), una implementazione open-source dei driver del Kinect basata su un reverse engineering dello

stesso. L'utilizzo di driver di terze parti ci renderebbe però impossibile utilizzare alcuni comodi tool presenti nel Kinect for Windows SDK.

Uno fra tutti, Kinect Studio: una comoda utility che permette di registrare delle sessioni di utilizzo dellìutente davanti al sensore e di poterle poi inviare in un secondo tempo all'applicativo che utilizza il sensore a fini di test e debug.



UNA SESSIONE REGISTRATA IN KINECT STUDIO

Si è preferito perciò utilizzare il componente ofxCommonBridge che fa da ponte verso i driver ufficiali del Kinect for Windows SDK anche se la procedura di installazione e configurazione del componente non è proprio immediata.

La spiegazione dettagliata è comunque disponibile all'indirizzo https://github.com/joshuajnoble/ofxKinectCommonBridge

Per poter usare il Kinect con l'applicativo è necessario installare il Kinect for Windows SDK v1.8

Capitolo 5
PASSI DI IMPLEMENTAZIONE

Vediamo ora i passi svolti per arrivare alla realizzazione dell'applicativo, i problemi incontrati e le soluzioni adottate per risolverli o aggirarli.

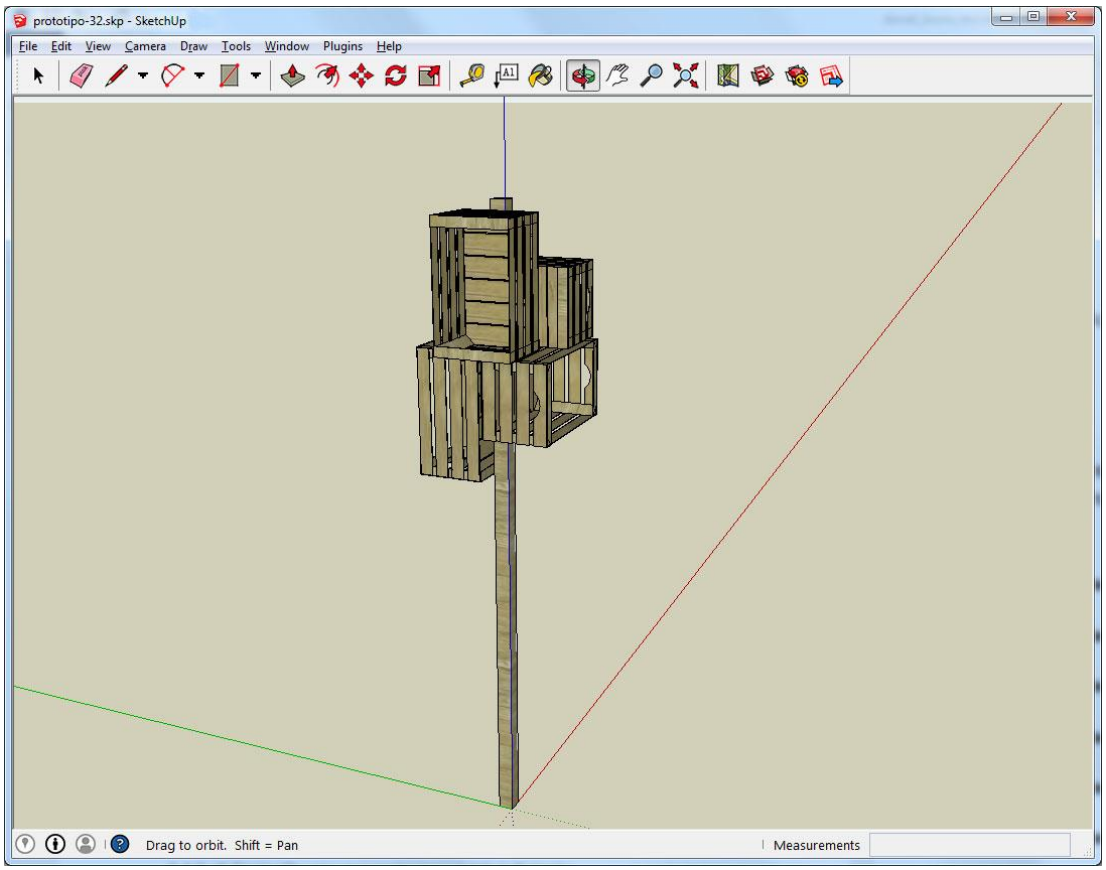
5.1 **Modellazione dell'oggetto da mappare**

Il primo passo da svolgere prima ancora di poter scrivere una riga di codice consiste nel modellare l'oggetto che vorremo poi andare a mappare. Non disponendo di uno scanner laser tridimensionale, ci si è armati di un metro, un programma di modellazione gratuito e un po' di pazienza. Per una buona riuscita della proiezione è importante che il modello sia il più possibile fedele alla realtà e che sia in scala con l'originale.

IL PROTOTIPO DA MODELLARE

5.1.1 *SketchUp*

E’ stato scelto come software di modellazione Sketchup, principalmente per la sua semplicità di utilizzo e per il fatto che è gratuito. Inoltre, il fatto che fosse un software già utilizzato dal collettivo di architetti, poteva tornare utile nel caso ci fosse stato da adattare il modello, in seguito a possibili modifiche effettuate sull’oggetto reale.



IL PROTOTIPO MODELLATO IN SKETCHUP

Questa scelta ha permesso di avere un modello pronto per l’uso in un tempo relativamente breve, ha però comportato una serie di imprevisti in seguito all’importazione che hanno richiesto l’utilizzo di appositi accorgimenti per limitare i problemi.

Terminata la modellazione si è esportato il modello in formato .3ds, in teoria pienamente compatibile con la libreria di importazione AssImp Model Loader utilizzata in OpenFrameworks.

5.2 **Importazione in OpenFrameworks**

Una volta ottenuto il file 3ds contenente la scena da gestire si è proceduto con l'importazione in OpenFrameworks.

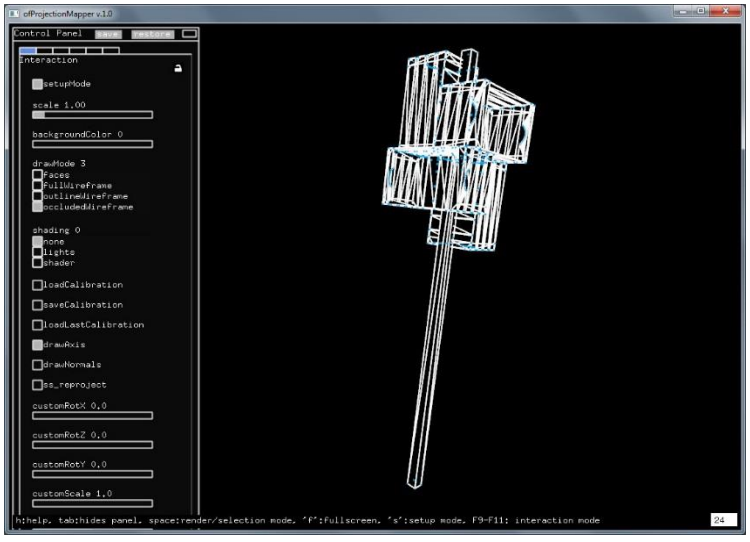
Si è utilizzato l'add-on ofxAssimpModelLoader, fondamentalmente un wrapper per la libreria open source "Open Asset Import Library" (http://assimp.sourceforge.net/)

5.2.1 *Assimp Model Loader*

Per caricare in memoria un modello è sufficiente istanziare la classe del loader e caricare il file 3ds

```
ofxAssimpModelLoader model;
model.loadModel("file_name.3DS", true);
```

Il metodo accetta un parametro booleano che applica, se impostato a vero, un'ottimizzazione del modello importato. Nello specifico, se l'ottimizzazione è attiva, gli oggetti che hanno lo stesso materiale associato vengono accorpati in un'unica mesh, altrimenti viene generata una mesh per ogni oggetto presente nel modello.



IL MODELLO CARICATO IN OPENFRAMEWORKS

Un problema riscontrato con il model loader è che eventuali gerarchie ad albero presenti in SketchUp vengono ignorate, per cui per poter gestire dei

gruppi (ad esempio per raggruppare gli elementi di una singola cassetta) sì è reso necessario un accorgimento: ad ogni cassetta è stato applicato un materiale con un nome differente, ma con la stessa texture. In questo modo non si notano particolari differenze a livello visivo, ma è possibile gestire a livello di codice le cassette come entità a se stanti.

5.3 **Il processo di calibrazione del sistema**

Per quanto riguarda la calibrazione del sistema ho utilizzato come base la procedura denominata “mapamok” a opera di Kyle McDonald, inizialmente rilasciata per la versione 0.7.4 di OpenFrameworks.

Le procedure sono state rese compatibili con la versione 0.8.3 con cui ho sviluppato l’applicativo e sono stati corretti dei bug legati al salvataggio e caricamento dei parametri di calibrazione e della mesh iniziale.

5.3.1 *L’approccio “mapamok” di Kyle McDonald*

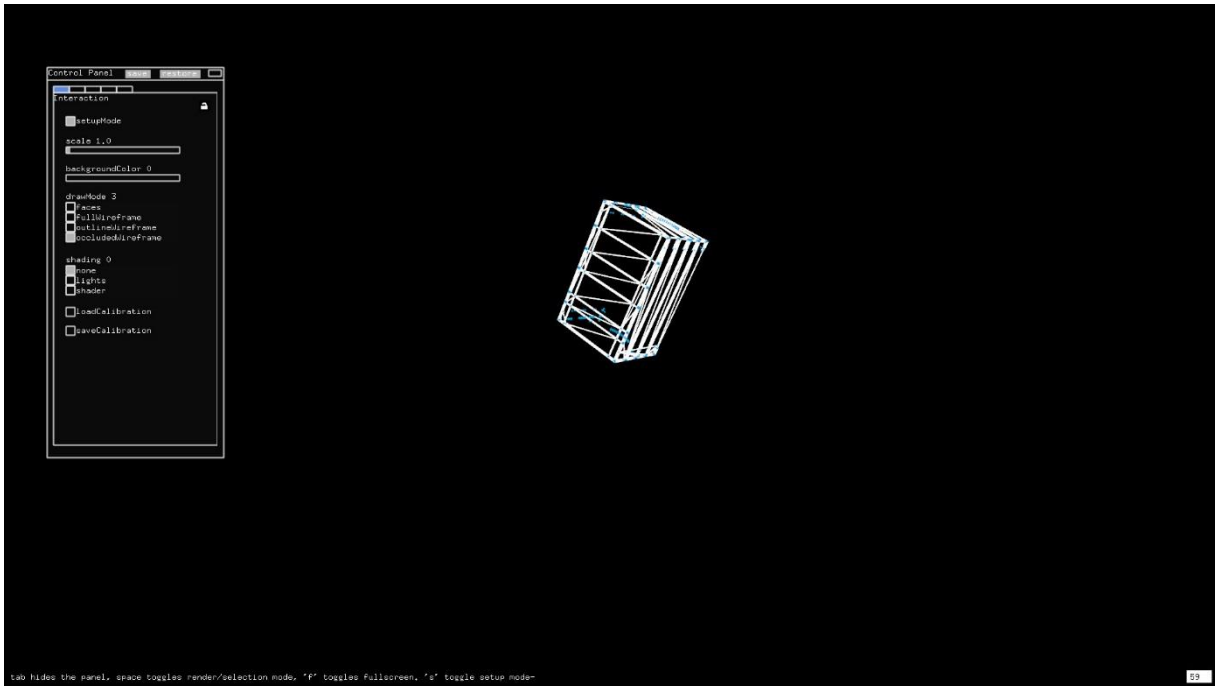
L’idea di fondo di mapamok è che la calibrazione del proiettore non dovrebbe portare via troppo tempo e che il meccanismo di proiezione funziona con un modello tridimensionale anziché con maschere bidimensionali.

Per fare ciò si prende in prestito il modello di calibrazione delle videocamere di OpenCV e lo si adatta alla natura del proiettore, ottenendo un tempo di calibrazione di qualche minuto. Alla fine del processo di calibrazione, i parametri vengono salvati su file. Alla successiva esecuzione del programma, se il proiettore e l’oggetto da mappare non sono stati spostati, è sufficiente caricare i parametri da file e la calibrazione è istantanea.

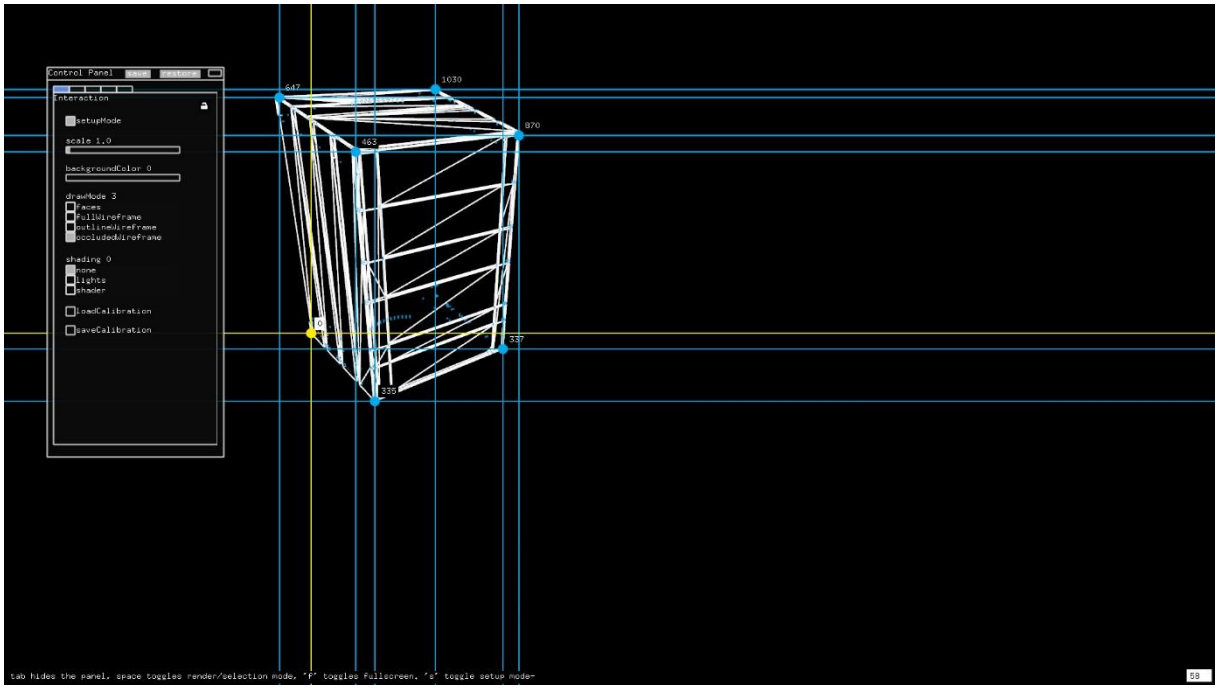
Il processo della prima calibrazione è piuttosto intuitivo e consiste fondamentalmente di tre passaggi:

1) Il caricamento del modello
2) La selezione di un punto da mappare in una visualizzazione libera che permette la rotazione della vista per facilitare l’operazione
3) Il trascinamento del vertice selezionato sul corrispondente vertice dell’oggetto reale in una modalità di visualizzazione dedicata.

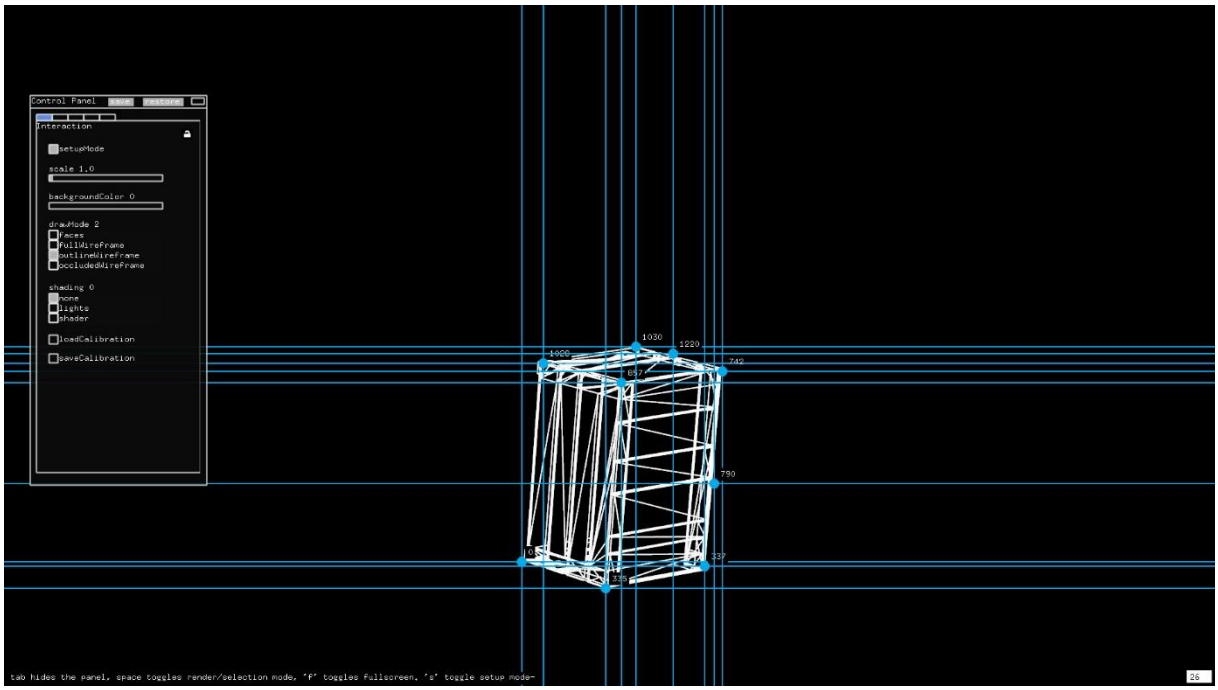
I passaggi 2 e 3 vanno ripetuti da un minimo di 6 a un massimo di 12 volte, già dal sesto punto è possibile visualizzare la mesh riproiettata.



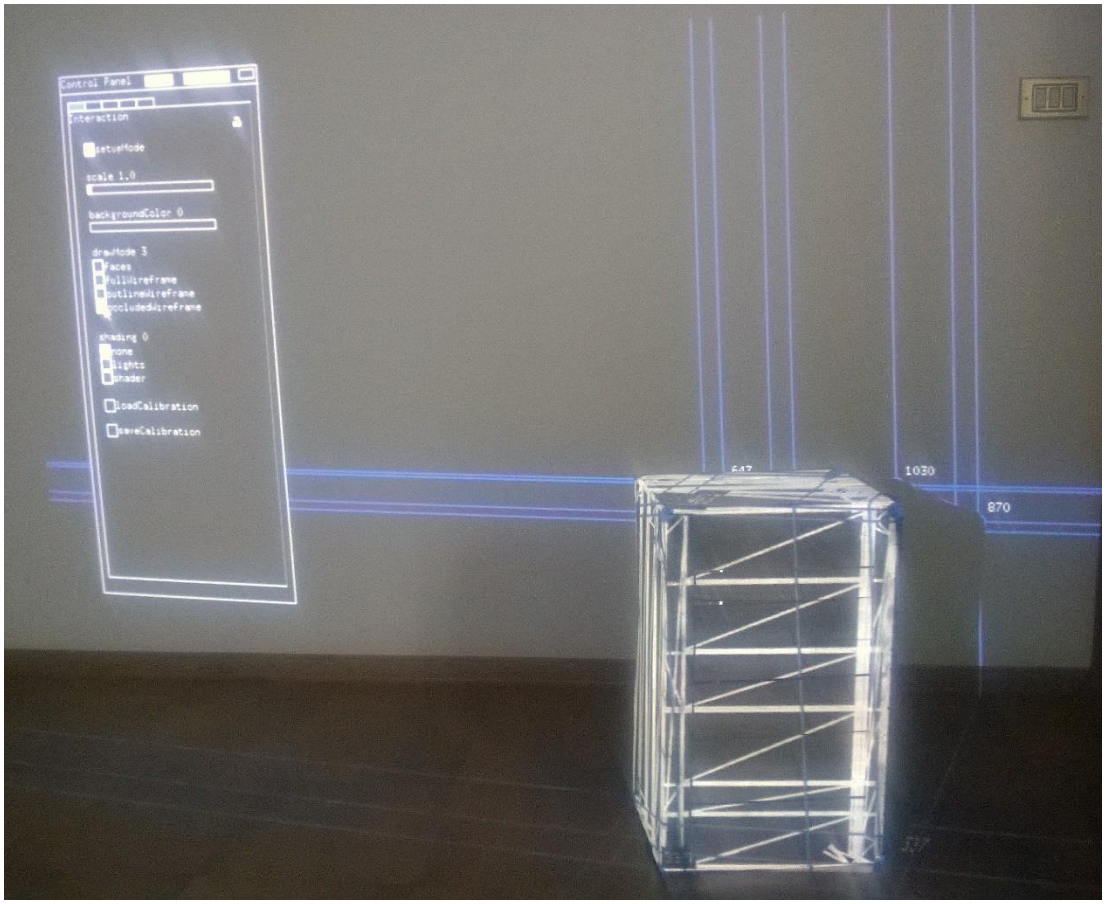
IL MODELLO CARICATO IN OPENFRAMEWORKS



LA MODALITÀ DI SELEZIONE VERTICE



LA MODALITÀ DI ASSEGNAZIONE VERTICE



IL RISULTATO FINALE

5.4 **Adattamento del modello**

Ora che il modello è caricato in memoria e il proiettore è calibrato per il mapping, dobbiamo decidere su quale parte del modello intervenire per modificarne alcune caratteristiche visive, come il colore o le texture. E' inoltre necessario impostare diversi parametri legati alla proiezione ma non codificabili all'interno del file .3ds. Si è reso perciò necessario gestire dei parametri impostabili facilmente dall'utente.

5.4.1 *ofxControlPanel e parametri impostabili dall'utente*

La soluzione scelta è stata di utilizzare l'add-on ofxControlPanel che permette con poche righe di codice di creare un pannello di controllo, che associa ad ogni controllo una variabile accessibile direttamente da codice.

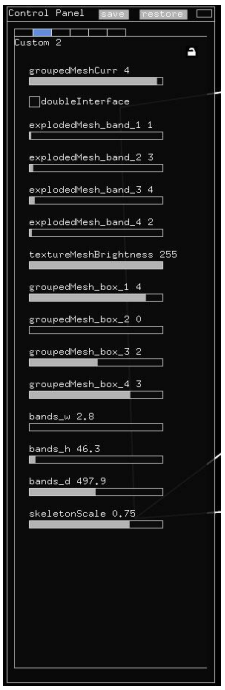
E' possibile modificare il valore della variabile a runtime agendo direttamente sui controlli. L'add-on fornisce anche una funzione di salvataggio e ripristino dei valori su file.

Con una semplice dichiarazione e poche chiamate di funzione

```
ofxAutoControlPanel panel;
...
panel.addPanel("Custom 2");
panel.addSlider("groupedMeshCurr", 0, 0, sceneManager.groupedMesh.size() -1,
true);
panel.addToggle("doubleInterface", false);
panel.addSlider("explodedMesh_band_1",0, 0, sceneManager.explodedMesh.size() -
1, true);
...
```

si ottiene un pannello di controllo per gestire dei parametri personalizzati. Si accede ai valori delle variabili associate ai controlli con delle semplici chiamate di funzione

```
sceneManager.groupedMeshColored = geti("groupedMeshCurr");
sceneManager.explodedMesh_band_1 = geti("explodedMesh_band_1");
```



IL PANNELLO DI CONTROLLO GENERATO DALL'ADD-ON OFXCONTROLPANEL

5.4.2 *Lo schermo secondario, una mesh aggiuntiva a runtime*

Una delle funzionalità richieste all'applicazione era di poter proiettare immagini relative al workshop e citazioni letterarie su sfondi fotografici.

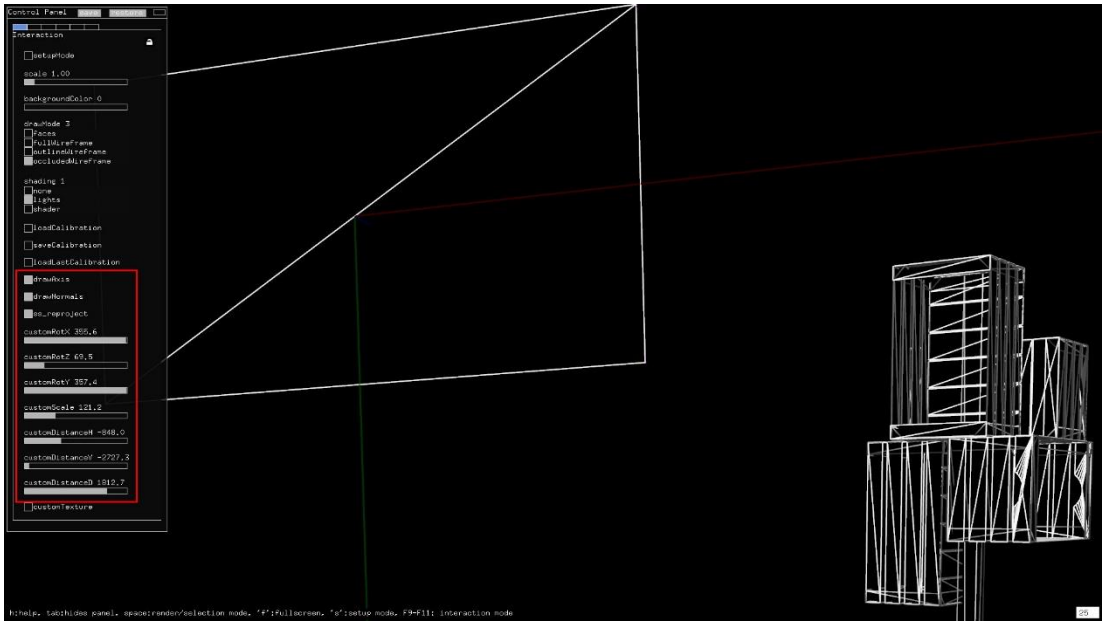
Dopo alcune prove è risultato che le dimensioni dell'albero di cassette della frutta non erano sufficientemente grandi per poter permettere una corretta visione, per cui vista la possibilità di sfruttare un muro nelle vicinanze dell'opera, si è deciso di implementare la gestione di uno schermo secondario virtuale, al cui interno venivano proiettate oltre alle foto e ai testi anche un'interfaccia grafica ad uso dell'utente finale.

Tutto ciò ha reso necessario risolvere il problema di calibrare la posizione del muro rispetto all'oggetto da mappare.

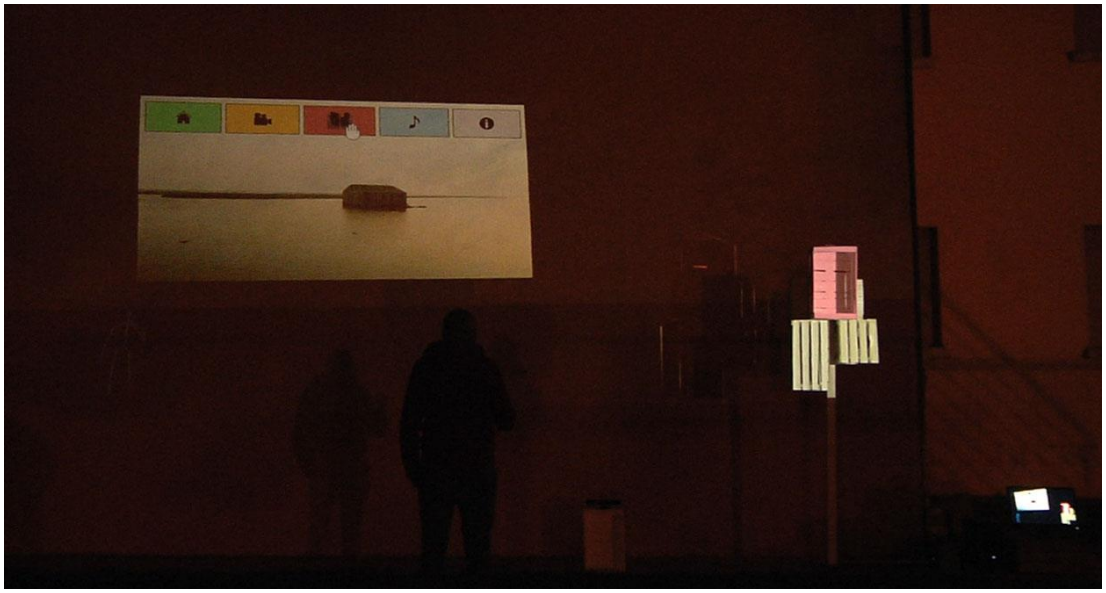
La posizione del muro non era nota a priori, per cui non era possibile inserirla direttamente nel modello 3ds, inoltre era molto probabile che una volta sul campo ci fossero da effettuare aggiustamenti e variazioni alla superficie dello schermo stesso.

Si è deciso di gestirlo come una mesh aggiunta alla scena a runtime tramite la classe ofPlanePrimitive e collocata nello spazio sfruttando dei parametri

inseriti dall'utente. I parametri consistevano nei valori di rotazione e traslazione sui tre assi a partire dall'origine della scena, più un fattore di scala per determinare la dimensione dello schermo di proiezione sul muro.



LO SCHERMO SECONDARIO E I RELATIVI PARAMETRI DI CONFIGURAZIONE



IL RISULTATO FINALE

5.4.3 *ofFbo, il Frame Buffer Object*

Un elemento chiave del progetto è rappresentato dal frame buffer object. Buona parte degli effetti video realizzati non possono essere disegnati direttamente su schermo perché risentirebbero della distorsione causata dalla posizione del proiettore non in asse con la superficie di proiezione.

Ogni effetto grafico deve perciò rientrare nella catena di trasformazioni applicate alla scena dopo la calibrazione.

Il modo più semplice per fare ciò è di agganciare una o più texture applicate ai modelli della scena e andare a disegnarci direttamente.

Per fare ciò si sfrutta un frame buffer object, che permette di allocare una parte di memoria video, e di modificarla utilizzando le stesse primitive che si userebbero per lo schermo principale.

```
ofFbo fbo;
fbo.allocate(w, h, GL_RGBA);
fbo.begin();
ofClear(255,255,255, 0);
ofBackground(0);
   ...
   // Istruzioni di disegno su schermo
   ...
fbo.end();
```

L'area di memoria viene poi utilizzata come sorgente per la texture associata all'oggetto.

Dichiaro perciò una texture, la faccio puntare al frame buffer object, collego la texture alla mesh già soggetta a proiezione e la disegno su schermo.

```
   ofTexture appTex;
   appTex = sceneManager.fboSideScreen.getTextureReference();
   appTex.bind();
   sideScreenPlane.drawFaces();
   appTex.unbind();
```

5.4.4 *Il problema delle coordinate delle texture importate*

Un problema piuttosto fastidioso e rimasto in parte irrisolto riguarda le coordinate delle texture applicate ai modelli in SketchUp. Sembra che nel processo di esportazione in 3ds da SketchUp e la successiva importazione in OpenFrameworks tramite la libreria ofxAssImpModelLoader le coordinate delle texture siano in un formato non corretto, per cui la tecnica precedentemente enunciata di sostituirle tramite frame buffer object non può essere applicata.

5.5 **Animazioni ed effetti**

In molte parti dell'applicazione sono presenti animazioni ed effetti dinamici relativamente alla posizione di pulsanti, dimensioni delle icone su rollover, movimenti e fade di immagini. Tutti questi effetti sono stati implementati da zero, basandosi su un add-on semplice ma potente, che permette di modificare dinamicamente il valore di una o più variabili nel tempo.

5.5.1 *ofxTween, la base di ogni animazione*

Il componente che sta alla base di buona parte delle animazioni presenti nell'applicativo si chiama ofxTween. Il componente implementa diverse funzioni di variazione dei valori nel tempo (lineare, quadratico, sinusoidale...) e permette di definire: il valore iniziale della variabile, il valore finale, la durata della variazione ed un eventuale delay iniziale. E' inoltre possibile testare se la variazione è ancora in corso o se è stata ultimata.

Esempio:

dichiaro l'oggetto tween e il tipo di funzione di variazione nell'header

```
ofxTween tweenlinear;
ofxEasingLinear easinglinear;
```

Imposto i parametri nel metodo setup() dell'applicazione

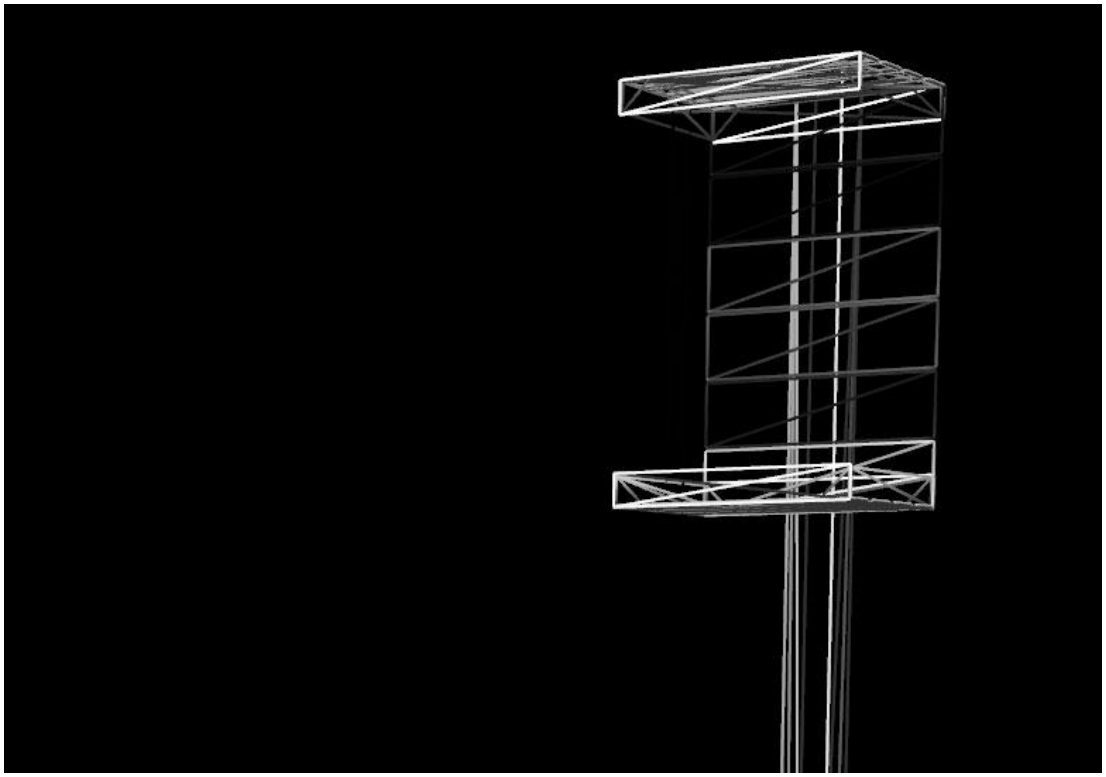
```
tweenlinear.setParameters(easinglinear, ofxTween::easeOut, 10, steps, duration,
delay);
```

Leggo il valore corrente nel metodo update() dell'applicazione

```
val = tweenlinear.update()
```

5.5.2 *L'animazione iniziale*

Un esempio pratico di utilizzo del componente ofxTween è l'animazione iniziale di disegno dell'albero di cassette.



UN FOTOGRAMMA DELL'ANIMAZIONE INIZIALE

Per realizzare l'animazione, è stato caricato in memoria il modello 3ds con l'opzione di ottimizzazione disabilitata in modo da avere in memoria una mesh per ogni asse di ogni cassetta.

E' stato poi allocato un vettore di tween per ogni mesh,

```
// inizializzo i tweens per ogni singola mesh
for (int i = 0; i < explodedMesh.size(); i++)
{
	ofxTween appTween;
	ofxEasingBounce appEasing;

	tweenAppareAlbero.push_back(appTween);
	easingAppareAlbero.push_back(appEasing);


}
```

Ad ogni tween è stato assegnato un intervallo da zero a 255 e un delay iniziale crescente.

```
int size = tweenAppareAlbero.size();

for (int i = 0; i < size; i++)
{

	litMesh[i] = false;

	tweenAppareAlbero[i].setParameters(easingAppareAlbero[i],
ofxTween::easeIn, 0, 255, tweenSingleDuration, 2000 +  (size - i - 1)  *
tweenTotDuration / explodedMesh.size());

}
```

Il valore del tween viene letto all'interno del metodo update() per impostare il valore di trasparenza della mesh corrente

```
for (int i = 0; i < tweenAppareAlbero.size(); i++)
{
	if (tweenAppareAlbero[i].isRunning()){
		alfaMesh[i] = tweenAppareAlbero[i].update();
	}
}
```

Il valore memorizzato viene poi utilizzato all'interno del metodo draw() per disegnare la mesh animata.

```
for (int i = 0; i < explodedMesh.size(); i++)
{
	ofSetColor(255, alfaMesh[i]);

	if (alfaMesh[i] != 0)
		explodedMesh[i].drawWireframe();

}
```

5.5.3 *Song player ed equalizzatore multibanda in tempo reale*

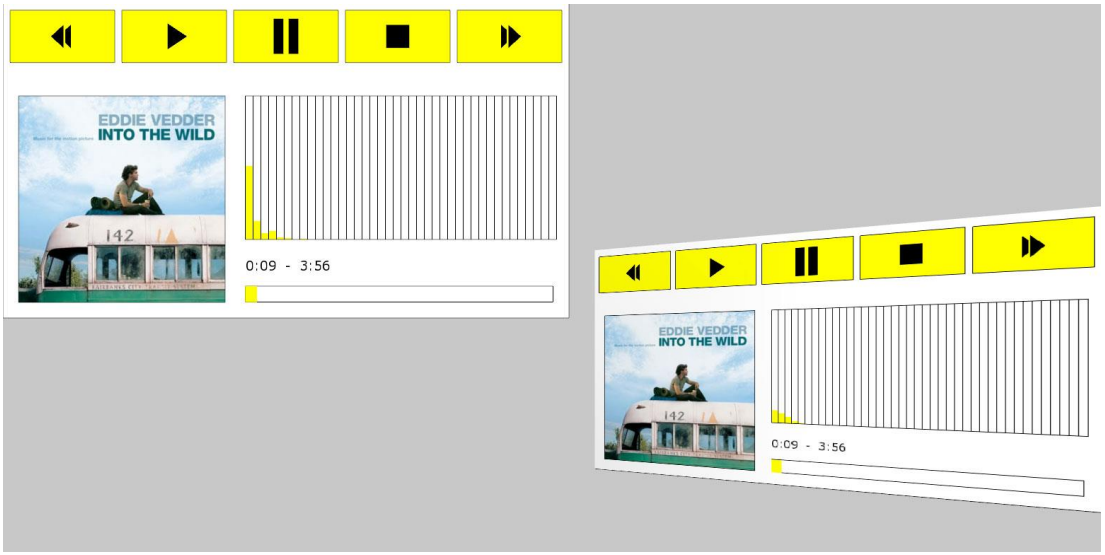
Una delle parte più interessanti da sviluppare dell'applicativo è sicuramente stata la realizzazione di un lettore musicale, sulla falsariga di un media player per cellulare.

OpenFrameworks fornisce una classe ofFmodSoundPlayer per la gestione dei file audio che permette di caricare in memoria un file wave o mp3 e di mandarlo in esecuzione tramite i classici comandi play, stop, pause.

La classe restituisce anche la durata del brano e lo stato di avanzamento corrente della riproduzione.

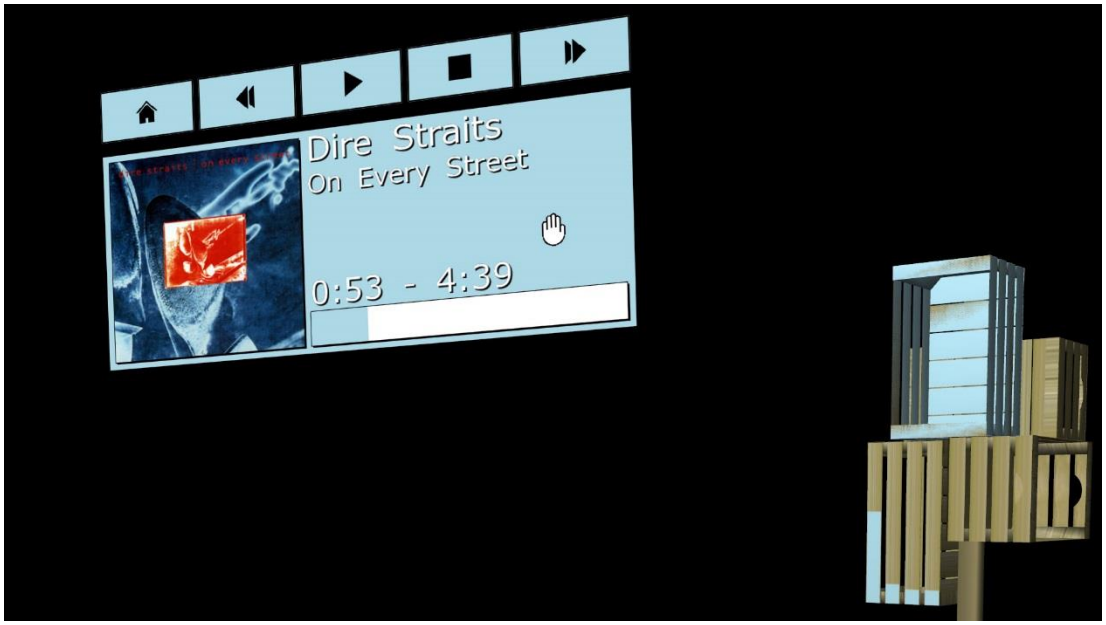
OpenFrameworks fornisce inoltre un interessante metodo ofSoundGetSpectrum(int nBands) che accetta in ingresso il numero di bande che vogliamo utilizzare nell'equalizzatore e ci restituisce un puntatore a un vettore di float contenente il relativo valore di ogni banda compreso fra 0 e 1.

Grazie a questi due strumenti e a un po' di olio di gomito è stato possibile realizzare un primo prototipo del lettore multimediale.



UN PROTOTIPO DEL LETTORE MUSICALE

Il lettore è stato poi migliorato ed ottimizzato per la proiezione finale; si è anche deciso di andare a rimappare l'equalizzatore sulle assi di una delle cassette della frutta.



LA VERSIONE FINALE DEL LETTORE MUSICALE

Capitolo 6
CLASSI SPECIALIZZATE DELL'APPLICATIVO

In questo capitolo andremo ad approfondire il codice di alcune classi specializzate, implementate per la gestione dei componenti delle varie scene interattive. Ogni classe è stata inizialmente implementata in un progetto separato in modo da testarne l'efficienza in un contesto più semplice che permettesse di effettuare il rendering direttamente sullo schermo.

Si è poi passato ad integrarle nel progetto principale dopo averle riconfigurate per renderizzare l'output un frame buffer object gestito dall'applicazione principale.

In genere le classi che fanno parte di OpenFrameworks hanno il nome che comincia con il prefisso of, mentre le classi che fanno parte degli add-on hanno il nome che comincia con ofx. Per evitare possibili conflitti nei namespace ho deciso di battezzare tutte le mie classi con il prefisso ofMy.

6.1 **Gestione delle Scene**

L'applicazione è suddivisa in diverse scene attivabili tramite l'interfaccia utente o da scorciatoie da tastiera. La gestione degli effetti da renderizzare nelle varie scene, il passaggio da una scena all'altra o la tipologia di input da utilizzare è gestito in una classe dedicata dal nome ofMySceneManager.

Ho definito nell'header degli alias relativi allo stato dell'input e alla scena corrente.

```
#define OFPM_STATE_SETUP 0
#define OFPM_STATE_INT_MOUSE 1
#define OFPM_STATE_INT_KINECT 2

#define OFPM_SCENE_SETUP 0
#define OFPM_SCENE_STANDBY 1
#define OFPM_SCENE_WORKSHOP 2
#define OFPM_SCENE_QUOTES 3
#define OFPM_SCENE_MUSICPLAYER 4
#define OFPM_SCENE_LIGHTS 5
```

```
#define OFPM_SCENE_CREDITS 6
#define OFPM_SCENE_SHOWON 7
#define OFPM_SCENE_SHOWOFF 8
#define OFPM_SCENE_HOME 9
```

e istanziato delle classi specializzate sviluppate appositamente per la gestione degli effetti audio e video

```
ofMyPictureSlider workshopSlider;
ofMySongPlayer workshopSongPlayer;

ofMyTextSlider citazioniSlider;
ofMySongPlayer citazioniSongPlayer;

ofMySongPlayerGui musicSongPlayerGUI;
ofMySongPlayer musicSongPlayer;

ofMyTextSlider creditiSlider;
ofMySongPlayer creditiSongPlayer;
```

un frame buffer object per la gestione delle texture sullo schermo secondario

```
ofFbo fboSideScreen;
```

dei metodi per l'inizializzazione e la riproduzione delle scene

```
void setup_IntroSlider();
void start_IntroSlider();

void setup_Workshop();
void start_Workshop();

void setup_Citazioni();
void start_Citazioni();

void setup_MusicPlayer();
void start_MusicPlayer();

void setup_Crediti();
void start_Crediti();
```

e un metodo per gestire gli eventi di click sull'interfaccia utente

```
void handleInterfaceClick(string & msg);
```

Per migliorare le prestazioni di ridisegno delle mesh, ho utilizzato due vettori di vboMesh, ovvero di mesh gestite come vertex buffer object, caricate direttamente nella memoria video.

```
vector<ofVboMesh> groupedMesh, explodedMesh;
```

Il modello in formato 3ds è stato caricato due volte in memoria, uno in modalità raggruppata per materiale e una con una mesh per oggetto. A seconda dell'effetto video che si vuole realizzare vengono disegnate le mesh di un vettore piuttosto che dell'altro.

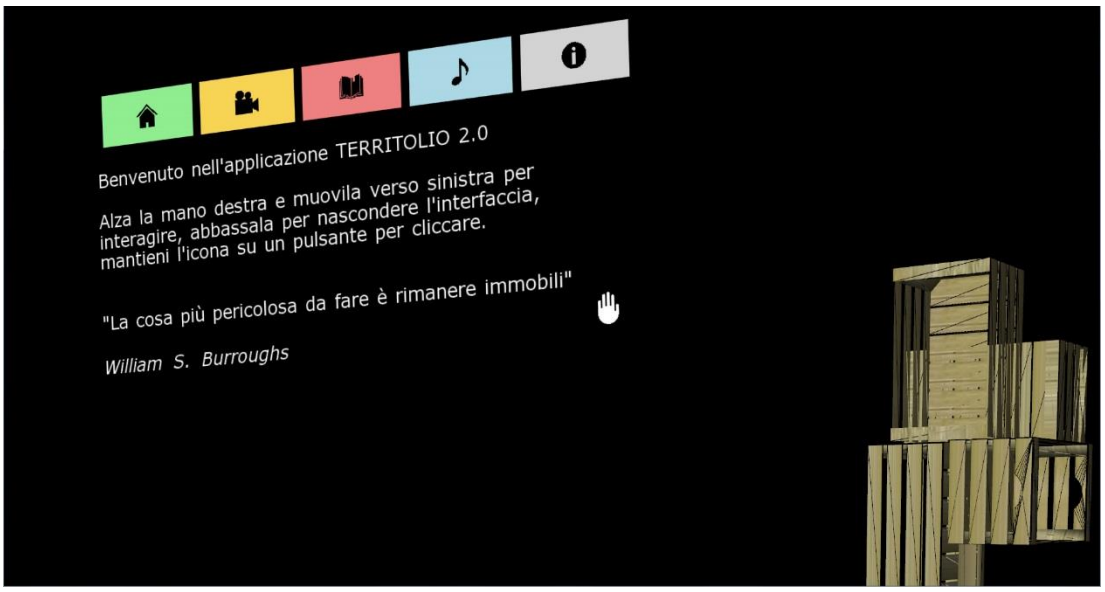
La classe espone anche due metodi pubblici

```
void setState(int st);
void setScene(int sc);
```

richiamati dalla classe principale per passare da una scena all'altra o per impostare il tipo di input

6.2 **L'interfaccia grafica**

L'interfaccia utente è stata resa il più semplice e intuitiva possibile per l'utente; a ogni pulsante è stata associata un'icona e un colore identificativo della scena corrispondente.

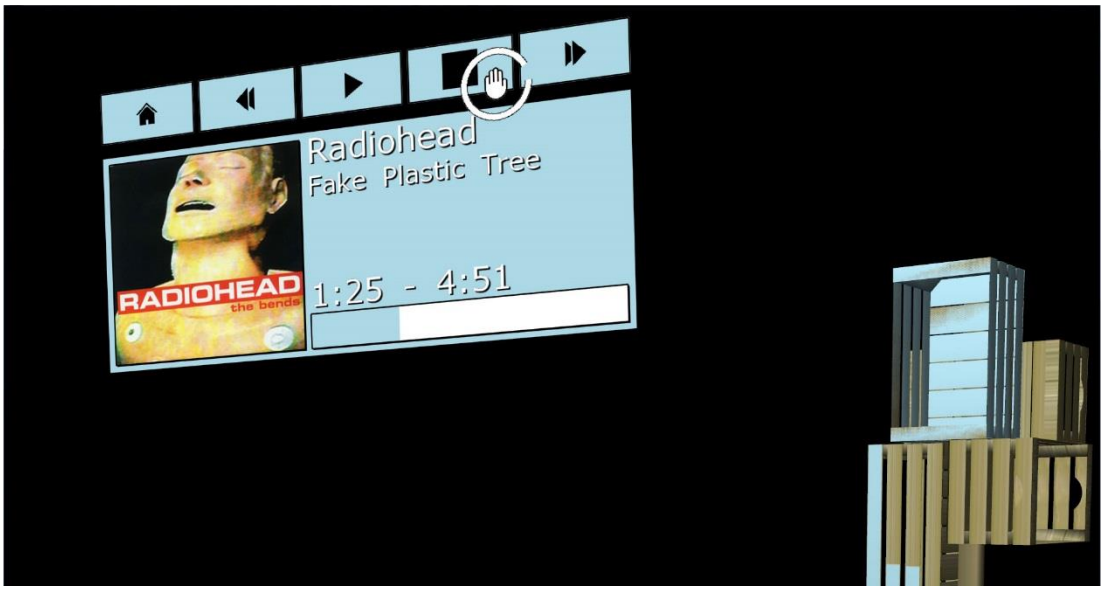


L'INTERFACCIA UTENTE CON LE ISTRUZIONI PER L'USO

Le icone sono state caricate in formato vettoriale svg tramite l'addon ofxSvg. Sull'evento di rollover vengono leggermente ingrandite per dare un feedback all'utente relativo al tasto con cui sta interagendo. Per confermare la sua scelta, l'utente deve mantenere il cursore sul pulsante per un tempo di circa 2

secondi. La pressione in corso viene mostrata all'utente tramite un cerchio disegnato attorno al cursore. Una volta completato il cerchio viene lanciato l'evento di click.

Tutti gli elementi dell'interfaccia ad eccezione delle immagini delle copertine dei dischi sono vettoriali, per minimizzare effetti di pixelizzazione legati alla distorsione prospettica della vista renderizzata.



L'INTERFACCIA GRAFICA CON L'EVENTO DI ROLLOVER

L'interfaccia viene visualizzata dinamicamente quando l'utente alza la mano destra (o muove il mouse nell'area in alto a sinistra dello schermo) e viene nascosta automaticamente non appena l'utente abbassa la mano (o sposta il mouse fuori dall'area sensibile).

Una volta entrati all'interno di una scena, la cassetta in alto viene colorata con il colore della scena stessa. In questo modo è possibile capire in che scena ci si trova anche se l'interfaccia utente non è visualizzata.

UN FOTOGRAMMA PRESO DALLA SCENA DEDICATA AL WORKSHOP

6.3 **Gestione delle Immagini**

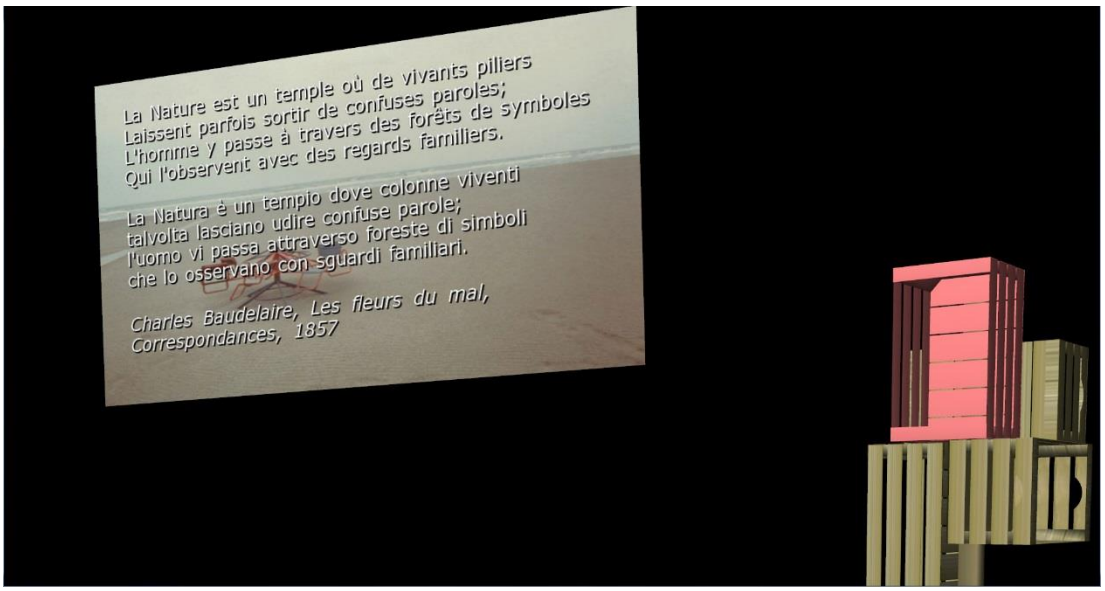
Le immagini visualizzate nelle varie scene sono gestite da una classe dedicata chiamata ofMyPictureSlider, la classe si occupa di caricarle e di visualizzarle temporizzandole con un effetto di pan e fade. Entrambi gli effetti sono gestiti combinando degli oggetti ofxTween con variabili legate alla posizione e alla trasparenza dell'immagine. Le immagini vengono disegnate su un frame buffer object inizializzato dallo scene manager e collocato come texture nella mesh dello schermo secondario.

I file delle immagini risiedono su disco e vengono caricate in memoria in base a un file di configurazione in formato xml.

6.4 **Gestione dei Testi**

I testi delle citazioni sono gestiti dalla classe ofMyTextSlider, la classe viene anch'essa inizializzata da un file xml esterno contenente i testi e le immagini corrispondenti da visualizzare in sottofondo.

I testi sono disegnati in formato vettoriale, grazie alla gestione nativa in OpenFrameworks dei font true type tramite la classe ofTrueTypeFont.



UN ESEMPIO DI TESTO CON IMMAGINE IN SOTTOFONDO

La visualizzazione avviene un carattere per volta, combinando l'oggetto ofxTween con la lunghezza della stringa da disegnare.

L'autore della citazione viene invece visualizzato alla fine del testo con un effetto di dissolvenza, combinando questa volta l'oggetto ofxTween con la trasparenza del testo.

L'effetto ombra è realizzato disegnando due volte lo stesso testo con colori differenti e uno scostamento in diagonale di un paio di pixel.

6.5 **Gestione dell'Audio**

L'audio delle varie scene viene gestito dalla classe ofMySongPlayer che viene anch'essa inizializzata da un file xml contenente l'elenco dei brani da riprodurre e i percorsi su disco dei file audio e delle copertine degli album.

Nel caso della scena del lettore musicale, oltre alla classe ofMySongPlayer, viene utilizzata la classe ofMySongPlayerGui che si occupa di disegnare su schermo le informazioni relative al brano e all'avanzamento dello stesso.

6.6 **Gestione dei Metadati**

Le differenti scene implementate hanno spesso alcuni aspetti in comune, ad esempio in quasi tutte le scene ci sono uno più brani musicali in sottofondo, ogni scena però ha delle musiche diverse dall'altra.

Un discorso analogo vale per le immagini dello slide show o per quelle utilizzate come sfondo alle citazioni letterarie.

Per ovviare a questo problema si è deciso di memorizzare le impostazioni di ogni scena su dei file esterni in formato xml, in questo modo è stato possibile utilizzare la stessa classe di gestione in scene differenti, limitandosi a modificare il file utilizzato dalla procedura di inizializzazione della stessa.

A titolo esemplificativo ecco un estratto del file di configurazione relativo alle citazioni letterarie (settings_citazioni.xml)

```
<citazioni>
  <quoteList>
    <quote>
      <text>
        \n\nKennst du das Land, wo die Zitronen blühn,\nIm dunklen Laub die
Goldorangen glühn,\nEin sanfter Wind vom blauen Himmel weht,\nDie Myrte still
und hoch der Lorbeer steht\n\nConosci tu il paese dove fioriscono i
limoni?\nNel verde fogliame splendono arance d'oro\nUn vento lieve spira dal
cielo azzurro\nTranquillo è il mirto, sereno l'alloro.\n
      </text>
      <author>J.W.Goethe, Wilhelm Meister Lehrjahre, 1795</author>
      <image>
        <file>citazioni\img\foto 2.jpg</file>
        <caption>caption 9</caption>
      </image>
    </quote>
  </quoteList>
</citazioni>
```

6.7 **La cartella Data**

Per comodità di utilizzo tutti i file utilizzati o generati dall'applicazione risiedono all'interno della sottocartella "data".

Possiamo trovarvi una o più cartelle nel formato "calibration-[data e ora del salvataggio" contenenti i file in cui vengono memorizzati i parametri di calibrazione del proiettore, un file settings.xml che contiene i parametri memorizzati dal pannello di controllo, una serie di file setting_[nome della scena].xml che contengono i testi e i percorsi dei file da utilizzare in ogni scena, e infine i file 3ds utilizzati dall'applicativo e i relativi file delle texture.

6.8 **Gestione dell'Input**

La gestione dell'input dell'utente viene svolta dalla classe ofMySceneManager. Nel caso si utilizzi come input il mouse, la classe si limita a leggere le coordinate del mouse e a passarle alla classe ofMyInterface che andrà poi a gestire gli eventi di rollover e di click.

Nel caso invece che l'input dell'utente sia ricevuto tramite il Kinect, entra in gioco una classe intermedia che si occupa di analizzare i dati ricevuti dal sensore.

La classe in questione si chiama ofMyKinectBridge e ad essa vengono passati gli scheletri generati dalla ofKinectCommonBridge che si occupa di collegare OpenFrameworks alle librerie Microsoft del Kinect.

Gli scheletri vengono analizzati e in base alla posizione della mano destra viene deciso se visualizzare o meno l'interfaccia utente e in che posizione mostrare il cursore.

Si è scelto come condizione di visualizzazione dell'interfaccia il fatto che la mano destra dell'utente sia più in alto del gomito destro. In questo modo per visualizzare o nascondere l'interfaccia è sufficiente alzare o abbassare la mano destra.

Per la gestione della posizione del cursore sullo schermo bisogna considerare che l'interfaccia deve funzionare con persone di differenti dimensioni e con diverse modalità di muovere la propria mano.

Ho deciso di tenere come riferimento la distanza della mano destro dal nodo della spina dorsale, sia per la componente orizzontale che per la componente verticale delle coordinate e di memorizzarne i valori minimi e massimi.

Ho poi utilizzato la funzione fornita da OpenFrameworks ofMap che permette di mappare un valore compreso fra due estremi su due nuovi estremi.

Prima di rimappare i valori li ho aumentati di un dieci percento, in modo da amplificare la portata del gesto e rendere meno difficoltoso il raggiungimento del bordo dello schermo.

In questo modo se l'utente aumenta il raggio d'azione della propria mano rimane comunque all'interno dell'area di interazione come si può notare nel seguente frammento di codice.

```
if (skel.size() > 0 ){

		thereIsSomeone = true;

		SkeletonBone rightHandBone = 
skel.find(NUI_SKELETON_POSITION_HAND_RIGHT)->second;
		if (rightHandBone.getTrackingState() == 
SkeletonBone::TrackingState::Tracked)
			rightHand = rightHandBone.getScreenPosition();

		SkeletonBone rightElbowBone = 
skel.find(NUI_SKELETON_POSITION_ELBOW_RIGHT)->second;
		if (rightElbowBone.getTrackingState() == 
SkeletonBone::TrackingState::Tracked)
			rightElbow = rightElbowBone.getScreenPosition();

		SkeletonBone spineBone = skel.find(NUI_SKELETON_POSITION_SPINE)-
>second;
		if (spineBone.getTrackingState() == 
SkeletonBone::TrackingState::Tracked)
			spine = spineBone.getScreenPosition();


		if ((rightHand.y < rightElbow.y)) { // || (rightHand.y < 
leftHip.y)){

			handStatus = MYKINECT_STATUS_ON;

			// DX
			dx = rightHand.x - spine.x;


			if (dx < minX)
				minX = dx;

			if (dx > maxX)
				maxX = dx;
```

```
                        cursorRelative.x = dx;

                        // DY

                        if (dy < minY)
                                minY = dy;

                        if (dy > maxY)
                                maxY = dy;


                        dy = rightHand.y - spine.y;


                        cursorRelative.y = dy;


                        // "Amplified clamp"
                        dx *= 1.1;
                        dy *= 1.1;


                        float appY = ofMap(dy, minY, maxY, 0, cursorArea.height,
true);

                        cursorPosition.x = ofMap(dx, minX, maxX, 0,
cursorArea.width, true);

                        cursorPosition.y = appY;


                } else {

                        handStatus = MYKINECT_STATUS_OFF;

                        initBounds();
                }


        } else {

                handStatus = MYKINECT_STATUS_OFF;
                thereIsSomeone = false;
        }
```

La procedura viene lanciato in ogni ciclo di update, in modo che i valori minimi e massimi dello spostamento della mano vengano aggiornati in tempo reale.

La classe ofMyKinectBridge si occupa inoltre di disegnare su schermo lo scheletro dell'utente nel caso venga richiesto dalla classe ofMySceneManager.

```
void ofMyKinectBridge::drawSkeleton(Skeleton sk)
{

	ofPushStyle();
	ofSetColor(ofColor::green);

	// per ogni joint, se tracked, recupero origine/destinazione e relative
screen position

	for (auto & bone : sk ){

		switch (bone.second.getTrackingState()){

		case SkeletonBone::Inferred:
			ofSetColor(0, 0, 255);
			break;
		case SkeletonBone::Tracked:
			ofSetColor(0, 255, 0);
			break;
		case SkeletonBone::NotTracked:
			ofSetColor(255, 0, 0);
			break;
		}

		auto index = bone.second.getStartJoint();
		auto source = sk.find( (_NUI_SKELETON_POSITION_INDEX) index);
		if (source != sk.end()) {

			ofLine(source->second.getScreenPosition(),
bone.second.getScreenPosition());

		}

	}


	ofSetColor(ofColor::blue);

	// draw joints
	for (auto & bone : sk){

		ofCircle(bone.second.getScreenPosition(), 5.0f);

	}

	ofPopStyle();

  }
```

Capitolo 7
UTILIZZO DELL'APPLICATIVO

Prima di utilizzare l'applicativo è necessario procedere alla calibrazione del proiettore, e dello schermo secondario di proiezione.

7.1 **Istruzioni per la calibrazione**

La parte di calibrazione funziona in due modalità: selezione e rendering. All'avvio dell'applicativo si è in modalità selezione, sulla sinistra è visibile il pannello di controllo, premendo il tasto TAB è possibile nasconderlo o mostrarlo.

Premendo il tasto SPAZIO si commuta tra la modalità di selezione dei vertici e la modalità di rendering della scena.

Per prima cosa dobbiamo scegliere i punti per calibrare il proiettore, ne servono un minimo di 6, generalmente 10-12 sono più che sufficienti.

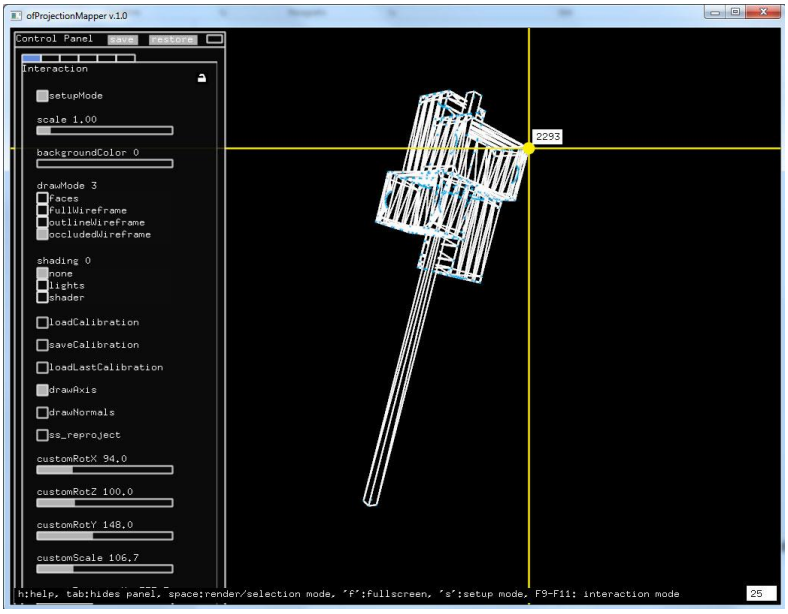
Iniziamo ad esaminare il modello ruotando la visuale premendo il TASTO SINISTRO del mouse e trascinando. Possiamo zoomare la visuale premendo il TASTO DESTRO del mouse e trascinando. Possiamo traslare la visuale tenendo premuto il tasto M sulla tastiera insieme al TASTO SINISTRO del mouse e trascinando.

Se spostiamo il mouse sopra uno dei vertici del modello, questo verrà evidenziato da una croce porpora e verrà visualizzato il numero del vertice.

Cliccando con il TASTO SINISTRO del Mouse possiamo selezionarlo, evidenziandolo in giallo, a questo punto possiamo passare alla modalità di rendering premendo il tasto SPAZIO per posizionare il vertice selezionato sulla corrispondente posizione dell'oggetto reale.

In modalità rendering possiamo spostare velocemente il vertice selezionato trascinandolo con il TASTO DESTRO del mouse, mentre per un

posizionamento più preciso possiamo trascinarlo con il TASTO SINISTRO del mouse. Ultimato il posizionamento possiamo tornare alla modalità di selezione premendo SPAZIO e ripetere la procedura per gli altri vertici.



SELEZIONE DI UN VERTICE PER LA CALIBRAZIONE

Dal sesto punto assegnato in poi dovrebbe essere visibile in modalità rendering l'oggetto riproiettato in base ai parametri di calibrazione calcolati.

Nel caso avessimo sbagliato ad assegnare la posizione a un vertice, possiamo eliminarla con il tasto BACKSPACE dopo aver selezionato il relativo vertice con il TASTO SINISTRO del mouse.

Per ottenere una buona calibrazione è consigliabile selezionare punti che coprano complessivamente l'intera scena, includendo punti al centro e ai bordi del modello. E' sconsigliato selezionare più punti vicini fra loro.

Terminato il processo di calibrazione è possibile salvare i parametri premendo il tasto saveCalibration nel pannello di controllo.

E' inoltre possibile nascondere le linee che evidenziano i punti di calibrazione premendo il tasto S

Altre utili scorciatoie da tastiera sono:

R per ricaricare gli ultimi parametri di calibrazione e i parametri del pannello di controllo.

F per passare dalla modalità in finestra a quella a schermo intero

Terminata la calibrazione del modello possiamo procedere a tarare lo schermo secondario. Per visualizzarlo bisogna attivare la voce ss_reproject nel pannello di controllo; è utile attivare anche la voce drawAxis. A questo punto si tratta di andare ad agire sugli slider customRot, customScale e customDistance per ridimensionare e collocare lo schermo secondario in una posizione consona all'ambiente in cui è si trova l'installazione.

E' possibile agire sugli slider del pannello di controllo con il TASTO SINISTRO del mouse per una variazione veloce e con il TASTO DESTRO per una variazione più fine.

Una volta ottenuta un'impostazione soddisfacente per lo schermo secondario occorre disabilitare l'opzione drawAxis e abilitare l'opzione customTexture.

Prima di passare alla modalità interattiva verificare che nel pannello di controllo il drawmode sia impostato a "faces", lo shading a "lights" e che le voci "ss_reproject" e "customTexture" siano abilitate.

Si consiglia inoltre di salvare i parametri settati nel pannello di controllo utilizzando il tasto "SAVE" presente nello stesso.

7.2 **Istruzioni per l'utilizzo interattivo**

Una volta terminata la calibrazione è sufficiente premere F2 per entrare in modalità interattiva e a scelta F11 o F12 per scegliere il tipo di input.

E' comunque possibile tornare alla modalità di calibrazione premendo F1

F11 abilita l'input dal mouse

F12 abilita l'input da Kinect

F10 disabilita l'input

K in caso di input da Kinect disegna/nasconde lo scheletro dell'utente

E' possibile bypassare l'interfaccia utente e saltare a una scena precisa premendo i tasti da F2 a F6. La pressione del tasto ESC provoca l'uscita dall'applicazione

7.3 **Personalizzazione dei contenuti**

Oltre ai parametri di calibrazione di proiettore e schermo secondario è presente una sezione parametri legati al file 3ds importato. Nello specifico si è salvato il numero della mesh corrispondente alla cassetta a cui cambiare colore in ogni scena, i numeri delle mesh corrispondenti alle singole assi a cui andare ad associare le bande dell'equalizzatore, la larghezza, altezza e profondità delle bande, la dimensione dello scheletro dell'utente da visualizzare su schermo.

PARAMETRI AGGIUNTIVI

E' inoltre possibile andare a modificare le immagini visualizzate, le citazioni e le musiche riprodotte, editando i file xml settings_*.xml presenti nella sottocartella data dell'applicativo.

CONCLUSIONI

La realizzazione di questo progetto è stata piuttosto impegnativa, ma mi ha dato parecchie soddisfazioni. Le potenzialità del projection mapping sono notevoli e realizzabili anche con strumenti relativamente economici. Buona parte di tutto ciò è però stato possibile solo grazie alla disponibilità di un framework open source che è tuttora in sviluppo e che merita di essere supportato. Spero che questo mio lavoro possa essere di ispirazione e che porti altre persone ad approfondire il tema del projection mapping.

RINGRAZIAMENTI

Un ringraziamento particolare al collettivo di architettura “La Prima Stanza” di Montiano e agli studenti del workshop “Vivere il Paesaggio” che hanno collaborato alla realizzazione del progetto.

BIBLIOGRAFIA E RIFERIMENTI

BIMBER Oliver, RASKAR Ramesh, *Spatial Augmented Reality, Merging Real and Virtual Worlds*, 2005.

MILGRAM Paul, KISHINO Fumio, *A Taxonomy of Mixed Reality Visual Displays*, Dicembre 1994 http://etclab.mie.utoronto.ca/people/paul_dir/IEICE94/ieice.html

PEREVALOV Denis, *Mastering OpenFrameworks: Creative Coding Demystified*, Packt Publishing, 2013.

BOUGUET Jean-Yves, *Camera Calibration Toolbox for Matlab*, Dicembre 2013 http://www.vision.caltech.edu/bouguetj/calib_doc/

Projection Mapping Central, http://projection-mapping.org/

OpenCV, Camera *Calibration and 3D Reconstruction* http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

MCDONALD Kyle, *Mapamok, an experimental projection-mapping tool*, Febbraio 2012 https://github.com/YCAMInterlab/ProCamToolkit/wiki/mapamok-%28English%29

OPENFRAMEWORKS, *Documentazione ufficiale* http://openframeworks.cc/documentation/