POLITECNICO DI TORINO
INGEGNERIA DELL'INFORMAZIONE

III Facoltà di Ingegneria
Corso di Laurea in Ingegneria delle Telecomunicazioni

Tesi di Laurea Magistrale

Ricostruzione grafica tridimensionale di un ambiente extraterrestre: la Luna

Creazione del Plastico Interattivo Digitale lunare

Relatori:
prof. Fulvio Corno
ing. Fulvio Dominici Carnino

Candidato:
Ettore Novarese

Luglio 2008

Ringraziamenti

Si ringrazia la Fondazione Ultramundum, in particolare nelle persone dell'ing. Fulvio Dominici Carnino e dell'ing. Lorenzo Bidone, per l'indispensabile supporto, sia umano sia tecnico, fornito nella realizzazione di questo lavoro, nonché per la disponibilità e la pazienza dimostrate.
Un ringraziamento speciale, però, va ai miei genitori che mi hanno sempre sostenuto, incoraggiato e spronato nel corso del lungo ed impervio cammino universitario, avendo fiducia in me e credendo nelle mie scelte, che oggi mi hanno portato all'ambita meta della laurea.

Indice

Obiettivi della tesi

Questo lavoro ha lo scopo di presentare un modello digitale tridimensionale della Luna. Tale progetto è stato realizzato utilizzando diversi tipi di risorse tra cui assumono un ruolo primario i dati della NASA, sia in forma di immagini sia di testo. Tuttavia, per poter utilizzare i file in nostro possesso, si è dovuto ideare e creare del software utilizzando il linguaggio di programmazione C/C++.
L'intero lavoro è stato realizzato presso la *Fondazione Ultramundum*, con l'auspicio di introdurre nuova linfa nel mondo multimediale e con la prospettiva di far conoscere, ad un pubblico sempre più vasto ed esigente, le potenzialità che questa Fondazione italiana porta con sé, la cui storia e progetti sono brevemente illustrati nella prima parte di questo documento.
Nella stesura del lavoro tesistico si è tenuto conto che il lettore fosse in possesso delle nozioni di base dell'informatica, del linguaggio C/C++ e dei rudimenti della grafica digitale. Viene di seguito riportato un breve compendio degli argomenti centrali sul quale verte la tesi in questione.

Ricerca e studio del materiale scientifico a disposizione

Primo passo di questa tesi è stato la ricerca di dati che potessero essere utili al raggiungimento dell'obiettivo finale della creazione del modello digitale. A tal proposito, si è rilevato di grande interesse, nonché ricco di informazioni utili, il sito della NASA da cui sono stati acquisiti i file delle immagini della superficie lunare.
Preso atto dello standard PDS a cui questi aderiscono, si è trovato la corrispettiva documentazione con cui interpretare i dati stessi, ma anche delle routine software per la loro gestione.

Acquisizione e traduzione dei dati per la creazione di un database di textures

Il passo seguente è stato quello di leggere i dati NASA, immagazzinati in certi tipi di formato troppo poco diffusi e di difficile manipolazione, e trasformarli in altri tipi più comuni. In particolare, secondo lo standard PDS, ogni immagine è immagazzinata in un file binario con estensione 'img', supportato da un numero assai esiguo di applicativi. Dunque, si è provvisto a trasformare i file dal tipo 'img' a quello 'bmp', uno dei formati a

più vasta diffusione e utilizzabile con un numero quasi infinito di programmi. Per tale fine, si è utilizzata la libreria di funzioni OAL, realizzata dalla NASA stessa, per la gestione dei file in formato PDS.
Una volta trasformati tutti i file, si è avuto a disposizione una collezione di immagini di tutta la superficie lunare, ovvero il database di textures da cui partire per creare il modello digitale tridimensionale.

Adattamento delle textures ai canoni più consoni per la creazione del modello digitale tridimensionale della Luna

Il sistema UltraPort, che gestisce e crea materialmente il modello digitale, necessita, in ingresso, di file con determinate caratteristiche, soprattutto per quanto concerne le dimensioni. L'ultimo importante passo, quindi, è stato quello di scrivere un programma per adattare le textures, così collezionate, alle effettive richieste del motore grafico, con il quale poi si riesce a creare l'intero plastico interattivo virtuale nel quale muoversi a proprio piacimento ed in totale libertà.

Capitolo 1

Introduzione

1.1 Fondazione Ultramundum

1.1.1 Ambito lavorativo

La Fondazione Ultramundum, ente *no-profit* italiano con sede in Grugliasco (Torino) all'interno del Parco Culturale "Le Serre", impegnato sin dalla sua nascita nel gennaio 2001 nella ricerca e nello sviluppo in ambito informatico, ha come obiettivo la diffusione della tecnologia UltraPeg brevettata, e in Europa e negli Stati Uniti d'America, dall'ingegnere Fulvio Dominici Carnino, attuale presidente della Fondazione medesima. Per maggiori informazioni si rimanda al sito internet di Ultramundum [1].
La tecnologia UltraPeg [cfr. paragrafo 1.1.2] si auspica diventi, in un futuro non troppo remoto, lo standard per la creazione e modellazione di ambienti virtuali tridimensionali, e basa la sua esistenza sulla definizione di una sorta di "scatola di costruzioni" virtuale, composta di "mattoncini intelligenti". Chiunque potrà creare un oggetto, o addirittura un intero ambiente, con il semplice assemblaggio di cotali mattoncini, in modo semplice ed efficace.
Ma se UltraPeg è la tecnologia a disposizione, UltraPort [cfr. paragrafo 1.1.3] è il sistema che ne consente l'effettivo utilizzo e che permetterà la diffusione della UltraVisione [cfr. paragrafo 1.1.4].

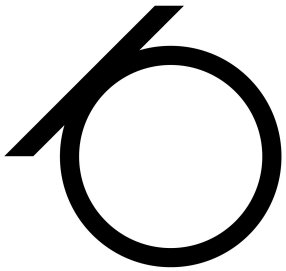

Figura 1.1. Alludrum, il logo di Ultramundum

1.1.2 La tecnologia UltraPeg

A livello di approccio, la sostanziale differenza rispetto alle tecnologie normalmente utilizzate per la realizzazione di ambienti tridimensionali interattivi risiede nella memorizzazione dei dati come concetti e non come collezioni di informazioni digitali, tecnica che fino ad oggi ha richiesto la fase di modellazione, assimilabile all'esigenza di dover scolpire un blocco di creta grezza per trarne i modelli di volta in volta necessari.
Tecnicamente parlando, si memorizza solo l'elenco dei numeri di serie degli elementi della scena e non ogni singolo dato. Così facendo si permette una più rapida velocità di trasmissione tra computer, per mezzo della rete Internet. Una nuova scena sarà quindi rappresentata da una collezione di codici, ognuno associato ad un preciso mattoncino.
Con UltraPeg, l'ambiente virtuale viene visto come una gigantesca scatola di costruzioni che offre una vasta gamma di pezzi diversi utili alla realizzazione della scena. In tal modo si evita all'utente di creare ogni volta un oggetto dell'ambiente, ma gli si offre la possibilità di sceglierlo fra i numerosi già esistenti e di modificarlo, se necessario, secondo le proprie esigenze, generandolo in modo semi-automatico impostando parametri diversi da quelli di base. Chiunque vorrà, potrà, ordunque, creare i mattoncini stessi che poi verranno resi disponibili a tutti, immettendoli nello scatolone virtuale delle costruzioni. I mattoncini elementari messi a disposizione da UltraPeg possono essere singoli elementi o aggregati molto complessi. Si possono prendere e usare interi edifici (come la Mole Antonelliana o Palazzo Madama), loro parti (porticati, arcate, portoni) o singoli elementi (fregi, capitelli, semplici mattoni).
In questo modo si potranno realizzare progetti evitando tutte quelle difficoltà che sino ad oggi avrebbero potuto ostacolarne l'effettiva realizzazione. La complessità degli elementi, le vaste estensioni areali, gli elevati costi di produzione, non saranno più un problema: più verrà sviluppato ed integrato lo scatolone, arricchendolo di nuovi pezzi, più ognuno di noi potrà creare i propri ambienti da rendere disponibile agli altri, in una sorta di grande comunità virtuale, in cui il realismo degli scenari consentirà un'esperienza più vicina alla realtà di quanto oggi sia possibile fare.
I mattoncini di base non si limitano ai soli modellini grafici tridimensionali, ma sono anche di tipo prettamente multimediale come musiche di sottofondo, animazioni e descrizioni testuali degli elementi per poter creare applicazioni più complesse ed immersive. Un banale ma significativo esempio è reso dai testi che descrivono parti degli ambienti: sono memorizzati in uno speciale formato che può essere tradotto automaticamente in ogni lingua del mondo per poi essere letto, da una voce narrante e sempre in maniera automatica, nel momento in cui l'utente lo desidera.
L'intelligenza dei mattoncini consiste nella capacità di adattarsi all'utilizzo che se ne vuole fare. Non essendo una collezione fissa di dati, bensì un programma vero e proprio, un porticato può, ad esempio, allungarsi o ridursi in modo da adattarsi al particolare contesto in cui si vuole inserirlo. Inoltre, un ambiente, essendo una serie di riferimenti agli elementi della raccolta di base, può migliorare nel tempo in modo automatico. Ad esempio, dopo aver creato una scena in cui appare il mattoncino della Mole Antonelliana, grazie alla memorizzazione del solo numero di serie e non dai suoi dati effettivi, in caso di successiva miglioria di tale mattoncino, un utente vedrebbe visualizzato l'elemento migliorato e non

quello utilizzato per la creazione della scena, avvenuta prima dell'aggiornamento. In pratica, ogniqualvolta viene aggiornato un mattoncino, automaticamente vengono aggiornate tutte le scene che ad esso fanno riferimento, senza che gli autori debbano fare alcunché.

1.1.3 Il sistema UltraPort

Tutte le scene create con la tecnolgia UltraPeg, vengono visualizzate per mezzo di UltraPort, il primo sistema operativo pensato appositamente per la grafica 3D. Semplicisticamente parlando, UltraPort è la parte client del sistema complessivo client-server che consente ad un "ultranauta" (colui che naviga nel mondo virtuale di Ultramundum) l'esplorazione dei mondi creati. In siffatta accezione, UltraPort può essere costituito, nel caso minimo, da un unico software che viene eseguito su un computer (con sistema operativo Windows, Linux, Macintosh, o altri) o da un opportuno decoder connesso ad un televisore (console per videogiochi, decoder tv appositi, o apparecchi analoghi).
Entrare nei dettagli di funzionamento di UltraPort esula dal nostro percorso formativo, ma in modo conciso, essendo un vero e proprio sistema operativo, possiede delle proprie entità, e quindi strutture dati, che ne consentono il funzionamento.
Tra tutte ricopre un ruolo da protagonista la *Tabula*, che altro non è che il nome tecnico dei mattoncini intelligenti. Ogni Tabula, identificata da un numero di serie univoco, è un file che può memorizzare al suo interno un elemento tridimensionale, un suono, un'animazione o un qualsiasi altro tipo di dato. Inoltre, ognuna di esse, può contenerne altre al suo interno.
Si arriva a capire che se una Tabula corrisponde ad un elemento, molte di esse formano un elemento più completo e complesso. Con questo sistema gerarchico di nidificazione e referenziazione delle entità, si comprende che ogni Tabula che rappresenta un oggetto di base (come, ad esempio, una ruota) è un elemento indipendente così come lo è la Tabula che rappresenta un'automobile, sebbene sia composta da altre Tabulæ. Inoltre, la ruota può essere usata anche per creare altre entità quali un trattore piuttosto che un autobus.
In pratica, ogni elemento rappresentato nell'ambiente 3D per mezzo delle Tabulæ, è sia elemento indipendente e compiuto sia entità di base per la creazione di oggetti più articolati.
Per creare una Tabula, UltraPort deve partire da altri dati più semplici che possono essere file standard di tipo testuale o grafico. In casi non del tutto rari ci si trova a dover utilizzare dei file che non corrispondono allo standard utilizzato dal sistema in questione. Per ovviare a tale problema si è pensato di creare un adattatore di file che prende il nome di UltraTools [cfr. paragrafo 1.1.5].

1.1.4 L'esperienza della UltraVisione

La missione della Fondazione è poter realizzare un nuovo tipo di televisione, denominato UltraVisione, che si basa proprio sulla tecnologia UltraPeg precedentemente descritta [cfr. paragrafo 1.1.2].
La UltraVisione altro non è che l'evoluzione tridimensionale della classica televisione e

permetterà a qualsiasi utente, utilizzando la rete Internet come mezzo diffusivo, di vivere esperienze alle volte difficilmente sperimentabili nella vita quotidiana.
Una struttura oramai consolidata, con canali ultravisivi disponibili sui server della Fondazione medesima, consentirà all'utente di fruire di servizi classici come documentari e film con la reale possibilità di interagire interattivamente seguendo il canonico punto di vista del regista ed introducendo, però, anche la concreta possibilità di immergersi ed aggirarsi liberamente, senza alcun vincolo preimposto, per l'ambiente sede del documentario o per il set del film, visitando, a piacimento, una riva di un lago visibile solo in lontananza o una via secondaria poco coinvolta dallo svolgimento delle vicende cinematografiche.
In ambito didattico si può sfruttare tale caratteristica di immersività nella realizzazione di una città. L'utente potrà visitarla seguendo il percorso classico, ma in ogni momento avrà, altresì, il controllo della telecamera e la possibilità di interagire con gli oggetti. In tal modo potrà svoltare in una via secondaria e chiedere informazioni su luoghi, edifici e monumenti con risposta vocale generata all'istante.
Nelle intenzioni della Fondazione vi è quella di creare un canale ultravisivo principale dal nome 4DGEA con scopi essenzialmente scientifici e didattici, nel quale verrà proposta una ricostruzione tridimensionale del mondo fisico che ci circonda, senza porsi alcun vincolo, se non nei limiti imposti dalla conoscenza umana, per poter rappresentare la realtà a partire dal dettaglio più ricercato fino ad arrivare all'intera galassia. Inoltre, esiste l'idea di affiancare a 4DGEA un altro canale suo parente stretto: 4DGEA+. Quest'ultimo avrà il compito di integrare i contenuti del primo, con ricostruzioni più ricche e spettacolari, eventualmente meno attendibili sul versante scientifico o addirittura ottenute in maniera procedurale qualora la disponibilità dei dati reali fosse qualitativamente o quantitativamente insufficiente, cercando, però, di attenersi il più possibile ai seppur esigui dati scientifici stessi.
In ambito videoludico le potenzialità della UltraVisione si concretizzano, ad esempio, nell'estensione illimitata degli spazi e nel grado di dettaglio degli elementi in scena arrivando, come esposto in precedenza, ad una precisione di livello scientifico, rendendo così il videogioco meno video e più gioco.
La natura *open-source* del progetto, rispettando i diritti di proprietà intellettuale ove richiesto, e la scelta del modello no-profit garantiscono, come è stato dimostrato in passato per altre tecnologie, una capillare diffusione sostenuta da un rapido sviluppo, ambo facilitati dalla comunità telematica che affida le proprie capacità alla comunicazione diretta tramite Internet. Inoltre, ci si protegge dalla possibilità di utilizzazione, da parte di terzi, di tale opera a fini di lucro, evitando, in tal modo, invidie e soprattutto problemi legali, che potrebbero portare all'arresto, o quanto meno al rallentamento, dello sviluppo dell'intero progetto.

1.1.5 La duttilità di UltraTools

UltraTools è un applicativo creato dalla Fondazione con cui è possibile creare ed importare i file nel sistema UltraPort adattandoli ai propri standard. Il software in questione consente di interpretare svariati formati di tipi di file per poi adattarli ai paradigmi e alle esigenze di UltraPort.

Un esempio che possa rendere chiara l'idea può essere rappresentato dai comuni file grafici presenti in tutto il panorama informatico. L'ampia quantità di formati esistenti ha portato a considerare il problema in termini di efficienza e di facilità di utilizzo in ambito Ultramundum: creare un intero sistema che gestisca un numero elevato di tipi standard di file grafici in continua evoluzione sarebbe stata un'impresa razionalmente concepibile ma praticamente irrealizzabile. Si è deciso, dunque, di far utilizzare al sistema UltraPort un sottoinsieme, di tutti i file grafici esistenti, composto dai più comuni come 'bmp' e 'jpg'.
L'esempio qui riportato si può estendere ad una schiera più corposa di file suddivisi per categorie. Il nodo centrale della questione è che serviva un gruppo base di file che il sistema potesse adoperare: la scelta, ovviamente, è ricaduta su un certo numero di tipi in modo da comprendere tutti i generi possibili, ma non tutti i formati.
Per ricondurci al lavoro di tesi, UltraTools permette, ad esempio, di acquisire i file immagini della NASA, realizzati secondo lo standard PDS [cfr. paragrafo 2.3.1] non comune, o comunque poco diffuso, e tradurli in file grafici di tipo 'bmp'.

1.1.6 Progetti

La Fondazione ha già realizzato una serie di complessi progetti per importanti istituzioni. Le possibilità di comunicare in modo nuovo ed estremamente convincente, con budget contenuti e con tempistiche brevi di realizzazione, ha convinto Comuni, Province, Regioni, Ministeri, Associazioni ed Imprese, che hanno utilizzato la tecnologia UltraPeg e hanno usufruito dell'esperienza di tutto il gruppo lavorativo (composto di esperti del settore informatico, ma anche di più semplici, ma non meno qualificati, collaboratori estemporanei come tesisti), per illustrare i propri progetti in maniera chiara, innovativa ed efficace. Il Comune di Torino, la Provincia di Torino, la Regione Valle d'Aosta, il Ministero dell'Istruzione e della Ricerca, il Politecnico di Torino, l'Associazione delle Piccole e Medie Imprese e altre realtà pubbliche e private, si sono dimostrate attente all'evoluzione della competenza messa a disposizione dalla Fondazione e ancora oggi hanno in alta considerazione il lavoro svolto, nonché la serietà e l'impegno profuso nei progetti conclusi, in quelli in corso d'opera e da qui a venire.
La Fondazione è divenuta partner del prestigioso progetto "Piemonte Virtuale" del Politecnico di Torino, finanziato dalla Regione Piemonte. Inoltre, ha partecipato e continua ad essere ospite gradito ed altamente stimato delle più importanti conferenze del settore di informatica grafica e realtà virtuale: "Virtuality", organizzata da Virtual Reality e MultiMedia Park; i congressi di MIMOS (Movimento Italiano Modellazione e Simulazione); "Imagina" a Montecarlo; lo SMAU a Milano. Presenziando a tali manifestazioni e presentando le proprie tecnologie e progetti, la Fondazione Ultramundum ha acquisito una sempre maggiore notorietà a livello internazionale.
Un caso emblematico delle potenzialità della Fondazione è la realizzazione di un prodotto chiamato PID (Plastico Interattivo Digitale). Grazie ad un forte impatto visivo della rappresentazione virtuale grafica ed interattiva, un PID illustra con immediatezza qualsiasi nuovo progetto, luogo o ricostruzione storica.
In occasione delle Olimpiadi Invernali Torino 2006, la Fondazione Ultramundum è stata chiamata ad affrontare un ruolo impegnativo, quale quello di promotrice e presentatrice

dei luoghi olimpici tramite la creazione del PID di Torino che, sebbene in maniera molto semplificata, rendeva il servizio per cui era stato pensato: visualizzare una pianta tridimensionale ed esplorabile della città di Torino con evidenziati i luoghi significativi e della città e delle olimpiadi stesse. Ma tale progetto non si è fermato, in quanto ancora oggi si cerca di migliorare tale plastico portando a compimento le creazione dettagliata di un numero sempre maggiore di zone di Torino come Piazza San Carlo e Parco Zappata, per come dovrà essere realizzato in futuro.
Tra i progetti in corso, alcuni assumono un ruolo chiave per il riconoscimento della bontà di tutto il progetto Ultramundum: la creazione del primo cartone animato tridimensionale open-source al mondo, il miglioramento del PID di Torino, la realizzazione del modello 3D della città di Roma, la ricostruzione dell'Italia in 3D e dell'intero Pianeta Terra in 4D, ovvero con la possibilità di viaggiare nel tempo. Si è pensato, però, anche ad ambienti extraterrestri.

1.2 Scopo della tesi

La tesi qui in discussione si incastona proprio nello scenario ad alto impatto visivo dello sviluppo di ambienti interattivi tridimensionali: la realizzazione del PID della Luna, che per quanto detto precedentemente, potrà essere considerato come un grande mattoncino intelligente con cui inscenare un film, un videogioco, un documentario o quant'altro possa scaturire dall'inventiva dei singoli che ne vorranno fare uso, trovandolo a disposizione nella scatola delle costruzioni.
Per riuscire a creare un tale ambizioso progetto c'è stata, in primis, la necessità di osservarsi attorno e riflettere attentamente su quali e quanti dati ci fossero effettivamente a disposizione. Successivamente si è operato sui dati medesimi per renderli consoni alla tecnologia UltraPeg.
Il primo passo, orbene, è stato quello di andare alla ricerca di dati scientifici apprezzabili. In tal caso, il sito della NASA (National Aeronautics and Space Administration) [2] è stato fonte inesauribile di informazioni, nozioni ed immagini, al contrario del sito della ESA (European Space Agency) [3] che non è stato del minimo aiuto in quanto l'agenzia non mette a disposizione i dati scientifici delle proprie missioni. I dati delle missioni NASA sono in numero così ragguardevole, che si è reso necessario un bel po' di tempo per capire dove indirizzarsi per ottenere il massimo di quanto a disposizione. Tra le tante effettuate da cui attingere dati, la migliore, per i nostri scopi, è stata la missione Clementine, da cui si sono ottenute e le immagini e le altimetrie della superficie lunare.
Una volta reperiti i dati, come descritto con maggiore dovizia successivamente, si sono dovuti creare dei "traduttori" di tipo software (anche se tecnicamente è più appropriato chiamarli "importatori") in modo che i dati stessi si potessero adattare alle esigenze e ai formati del cuore grafico e computazionale del sistema UltraPeg.
Dopo aver tradotto le immagini, si è capito che il metodo migliore per creare il PID lunare fosse quello di realizzare una sorta di gigantesco mosaico, grazie proprio alla collezione di tali immagini, per poterlo, così, applicare su un geoide puramente geometrico, in modo da avere la rappresentazione tridimensionale della Luna.

Analogamente, le altimetrie di valli, crateri e di tutta la superficie in generale, si è pensato di tradurle per poterle utilizzare con UltraPeg fino ad inserirle nel contesto grafico e creare il modello virtuale, in modo da dare all'intera ricostruzione quella sensazione di reale passeggiata lunare, con la possibilità di cadere in un cratere o salire su un colle. Purtroppo si è anche constatata la difficile integrazione dei dati altimetrici con l'intero sistema UltraPort, fino ad abortire l'idea, almeno per il momento, di utilizzo di tali dati.

1.3 Situazione attuale

1.3.1 Situazione consolidata

Nell'ambito delle realizzazioni grafiche, in giro per la rete Internet si possono trovare non tantissimi programmi in grado di dare una rappresentazione veritiera della Luna. Tra questo esiguo numero, va sicuramente menzionato "Celestia" [4]: una simulazione spaziale in tempo reale che permette di fluttuare nell'intero nostro sistema solare in un ambiente tridimensionale e che maggiormente si avvicina alla realtà scientifica dei fatti e delle osservazioni.
Questo programma, sebbene fatto con la dovuta attenzione e minuziosità dei particolari, ha il difetto di non offrire all'utente la discesa sul pianeta o sul satellite che si vuol visitare. In pratica, il dettaglio grafico di quanto proposto, non consente di poter effettuare passeggiate con la reale sensazione di essere atterrati nel luogo di nostro interesse: la definizione delle immagini risulta scarsa e l'esperienza virtuale della nostra gita risulta non essere più gradevole, proprio perché immersi in un mondo fatto di pixel esageratamente evidenti.
Un altro programma abbastanza curato è "World Wind" realizzato dalla NASA stessa e reperibile nel proprio sito [5]. Purtroppo per loro, per la realizzazione della Luna hanno utilizzato le immagini del database della missione Clementine, senza tenere conto della presenza di numerose zone in cui l'assenza di dati porta alla visualizzazione di intere aree nere, anche abbastanza estese, come si nota dalla figura 1.2.

1.3.2 Obiettivi

Proprio l'analisi dei difetti di Celestia e di World Wind, ha spinto le nostre forze nella direzione intrapresa: ricercare immagini con dettaglio maggiormente fine e dati più precisi e affidabili, con lo scopo, tuttavia, di creare un vero plastico interattivo, in cui si possa vivere un'esperienza realmente immersiva, con immagini il più possibile definite e complete.
È facile, altresì, capire che la Fondazione Ultramundum, notando un così poco realismo planetario, abbia deciso di infondere nuova linfa nel panorama della rappresentazione spaziale, conscia di avere i mezzi e la voglia di raggiungere lo scopo di una virtualità immersiva, il più possibile reale. Senza trascurare il fatto che in un futuro, sempre in dipendenza della disponibilità di dati precisi, si possa ricreare l'intero sistema solare in modo scientificamente corretto così da poter simulare dei veri e propri viaggi interplanetari col solo ausilio di un personal computer e di un collegamento Internet.
Bisogna tenere presente, difatti, che la Fondazione sta già lavorando al progetto del PID

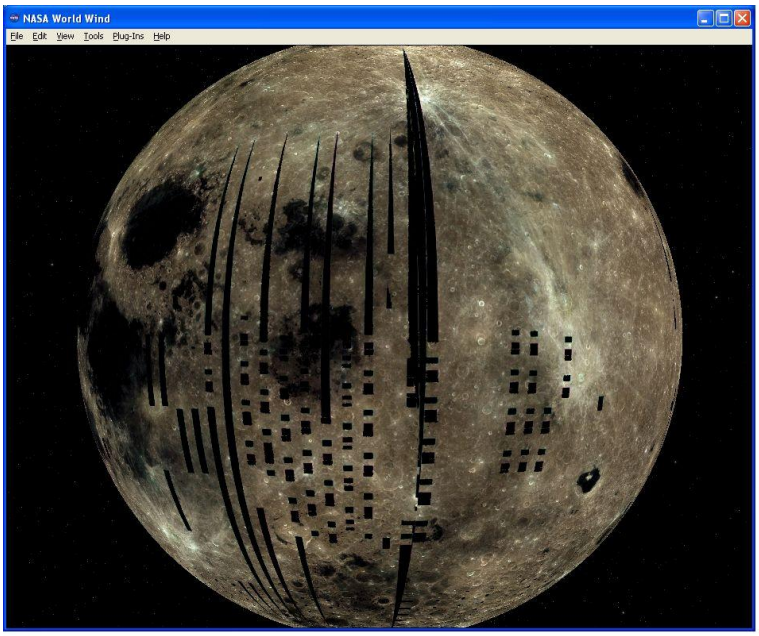

Figura 1.2. Visualizzazione della Luna con World Wind della NASA

della Terra, cercando di creare una versione 3D del più famoso, ma bidimensionale, "Google Earth" [6]. Pezzo per pezzo, pianeta per pianeta, si potrà avere l'intero sistema solare e poter realizzare, così, un planetario virtualmente reale.

1.3.3 Confronto con le altre realtà di ricerca

Per quanto concerne il PID della Luna, va detto che già oggi una delle applicazioni che è possibile trovare in rete è assai poco realistica e bassamente dettagliata: si tratta di "Google Moon" [7], che attualmente è una misera versione 2D di una sorta di mappa della superficie lunare, che utilizza una collezione di immagini a risoluzione non elevata, resa disponibile dalla NASA.
Attualmente, però, è noto che proprio la squadra di sviluppatori di "Google" sta lavorando al progetto ambizioso della ricostruzione grafica in 3D della Luna, ovvero il PID lunare che la Fondazione Ultramundum presenta con questa tesi.
Va precisato che se Google può vantare la stretta collaborazione con gli ingegneri della NASA, noi, qui in Ultramundum, abbiamo dovuto affrontare le difficoltà con le sole nostre forze in quanto, alle richieste di dettagli e spiegazioni ai competenti tecnici della NASA, non abbiamo ricevuto alcuna risposta.
Inoltre, è chiaro precisare sin da ora, che per stessa ammissione degli ingegneri della NASA, i set di immagini e di altimetrie su cui gli sviluppatori Google stanno lavorando, ma anche da noi utilizzati, sono di pessima fattura, sebbene risultino effettivamente i migliori a disposizione in termini di attinenza scientifica e grado di soddisfazione visiva.

Capitolo 2

Plastico Interattivo Digitale della Luna

2.1 Introduzione

Il PID (Plastico Interattivo Digitale) della Luna è una ricostruzione grafica tridimensionale dell'ambiente lunare, atta a poter dare la possibilità di esplorare un luogo di altrimenti difficile raggiungimento. Tale progetto, come esposto in precedenza, si inserisce nel contesto più esteso del mondo della UltraVisione [cfr. paragrafo 1.1.4] portando, quindi, nelle case di chiunque possieda una connessione Internet veloce, la reale possibilità di potersi fare una passeggiata dove l'astronauta americano Neil Alden Armstrong per primo, nel 20 luglio 1969 dopo l'allunaggio del modulo lunare *Eagle* della missione *Apollo 11*, fece «un piccolo passo per un uomo, ma un grande balzo per l'umanità».
Il PID vuole essere non solo un applicativo a sé stante, basato sulla tecnologia UltraPeg, bensì l'inizio di una serie di mattoncini intelligenti per la creazione di ambienti spaziali 3D. Con tale mattoncino si potranno creare documentari riguardanti le varie missioni che hanno portato l'uomo a sbarcare sulla Luna, sceneggiare film ivi ambientati, nonché collocare un videogioco proprio nello scenario lunare.
La ricostruzione grafica si basa sui dati ufficiali della missione *Clementine*, messi a disposizione dalla NASA sul sito Internet relativo alla missione stessa [8] e comprende file grafici e di testo appositamente strutturati e codificati secondo lo standard PDS (Planetary Data System). Si può intuire facilmente che per poter manipolare tali tipi di file è stato necessario lo studio dello standard e dei file medesimi, nonché la capacità di metterli in realzione ai tipi di dati a più larga diffusione.
Si pone in evidenza che i dati della NASA, raccolti durante le missioni che negli anni si sono susseguite per poter conoscere e studiare lo spazio extraterrestre, hanno la più grande rilevanza scientifica in ambito aerospaziale. Detto ciò, si desume che anche il PID lunare, poiché direttamente derivato e creato da essi, abbia un valore di base più vicino alla scienza di quanto altri prodotti possano, al giorno d'oggi, reclamare o pubblicizzare.
Per poter ottenere il modello grafico lunare così come presentato in questo lavoro di tesi, si è dovuto modificare e riorganizzare l'applicativo UltraTools già esistente, integrandolo con

nuove funzionalità. In particolare si sono dovuti creare due programmi che realizzassero la conversione dei file da un certo tipo a quello utilizzato dal sistema UltraPort.
Tramite la programmazione in linguaggio C/C++ si sono dunque realizzati:

- un importatore che traducesse i file grafici di tipo 'img' della NASA nei più comuni file di tipo 'bmp', creando così una collezione completa di immagini tale da coprire la superficie lunare nella sua interezza [cfr. capitolo 3];
- un adattatore che, partendo da questa libreria di immagini, creasse tanti piccoli tasselli di forma regolare in modo da adattarsi allo standard dei dati di cui UltraPort necessita [cfr. capitolo 6];

I file di dati della NASA aderiscono al formato PDS. La gestione di tali file nel formato nativo PDS è stato possibile grazie all'esistenza di una collezione di metodi creata appositamente dai tecnici NASA: la libreria OAL (Object Access Library).

2.2 La missione *Clementine*

2.2.1 Breve storia

La missione Clementine, ufficialmente denominata DSPSE (Deep Space Program Science Experiment) è stata la prima di una serie di dimostrazioni tecnologiche organizzate unitamente dal reparto BMDO (Ballistic Missile Defense Organization) del dipartimento DOD (Department Of Defense) e dalla NASA.
DSPSE fu lanciata il 25 gennaio 1994 a bordo del razzo Titan IIG dalla base dell'aviazione militare di Vandenburg in California. La missione prevedeva due mesi di mappatura sistematica della superficie della Luna e la ricognizione dell'asteroide terrestre "Geographos". Un errore nel software di bordo, in combinazione alle estreme condizioni a cui era sottoposto l'intero apparato hardware, portò, il 7 maggio 1994, all'imprevisto avvitamento della navetta con conseguente perdita dei gas dal sistema propulsivo di controllo. Tutto ciò precluse solo il secondo dei due obiettivi: lo studio di Geographos. La navetta venne affettuosamente soprannominata Clementine quando, una volta terminata la sua breve e incompleta missione, si perse per sempre, come in una nota canzone omonima.
L'obiettivo primario era l'affinamento dei sensori spettrometrici e di tutte quelle tecnologie utili per le generazioni successive di navette del DOD: il ricognitore satellitare, l'unità di misurazione inerziale, le ruote reattive, le batterie a nickel-idrogeno e i pannelli solari. Il DSPSE rappresentava una nuova classe di navette spaziali piccole, a basso costo e con enormi potenzialità, in grado di supportare e utilizzare le nascenti tecnologie spettrometriche essenziali per tutta una serie di missioni a lungo termine nello spazio profondo. Il secondo obiettivo era l'acquisizione di dati su Luna e Geographos per l'interà comunità scientifca e civile.
Clementine, infatti, contava di portare a compimento sette diversi esperimenti con altrettante tecnologie di acquisizione di dati: una camera UVVIS (UltraViolet/VISible), una camera NIR (Near InfraRed), una camera LWIR (Long Wavelength InfraRed), una camera HIRES (HIgh RESolution), due camere Star Tracker, un sistema LIDAR (Laser Image Detection And Ranging system).

2.2.2 Tecnologie e strategie di acquisizione dati

Nella molteplicità delle sperimentazioni effettuatesi con la missione Clementine e nel contesto del lavoro qui proposto, assumono un'importanza relativamente maggiore alcune delle tecnologie presenti sulla navicella: si tratta, in particolare, della camera UVVIS e del sistema LIDAR che hanno consentito di acquisire quei dati lunari che si sono rilevati di fondamentale importanza per la realizzazione del PID della Luna.

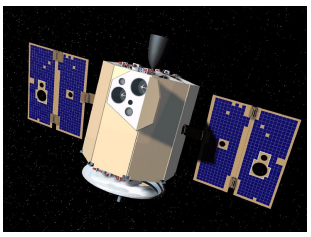

Figura 2.1. La navicella Clementine

La camera UVVIS

La camera UVVIS è un sensore composto di un telescopio catadiottrico e di lenti al silicio fuso montate su un dispositivo CCD (Charge Couple Device) per immagini digitali.
Il suo compito era di acquisire immagini della superficie della Luna con il dovere, per avere una collezione di dati più ampia e completa, di utilizzare ben cinque bande spettrometriche: 415 nm (con una larghezza di banda di 40 nm), 750 nm (10 nm), 900 nm (30 nm), 950 nm (30 nm), 1000 nm (30 nm). Come si nota anche dalla denominazione della camera stessa, lo spettro di acquisizione varia dalle frequenze dell'ultravioletto fino a quelle dell'infrarosso, passando per tutto lo spettro visibile all'occhio umano.
Questo esperimento ha prodotto informazioni sulle proprietà minerarie e sul materiale superficiale della crosta lunare, oltre che dare immagini utili per gli studi morfologici, molte delle quali, però, poco utili a quest'ultimo scopo, in quanto prese con il sole basso. Le immagini hanno una risoluzione che varia dai 100 ai 400 metri/pixel, in dipendenza della posizione della navetta.

Il sistema LIDAR

Il sistema LIDAR è stato concepito per misurare la distanza tra la navicella e un qualsiasi punto della superficie lunare, così da poter creare una mappa altimetrica da poter essere utilizzata per studiare sia la morfologia del terreno lunare sia le proprietà della litosfera, fino a, combinando tali dati con quelli gravitazionali, poter ricavare un quadro completo sulla distribuzione della densità della crosta.
L'intero sistema altro non è che un trasmettitore laser a 1064 nm di banda e condivide l'apparato elettromeccanico con la camera HIRES. Il laser invia impulsi sulla superficie lunare per poi, una volta riflessi dalla superfice e acquisiti, calcolare, in base al tempo

impiegato per andare e tornare, il valore altimetrico del punto preso in esame. Sebbene abbia un intervallo operativo nominale di 500 km, sono state acquisite altitudini anche oltre i 640 km.

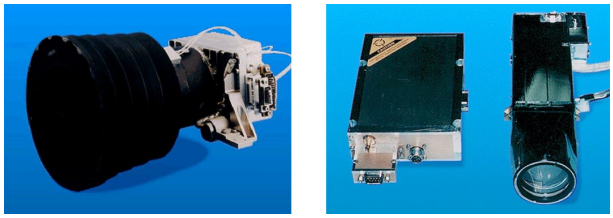

Figura 2.2. La camera UVVIS e il sistema LIDAR

La camera HIRES

La camera HIRES è composta di un telescopio con un intensificatore di immagini con tecnologia CCD. Il sistema è stato progettato per acquisire ampie porzioni selezionate della superficie lunare. Le informazioni acquisite hanno consentito lo studio dei processi morfologici e, in combinazione con i dati spettromagnetici, della composizione geologica e mineraria della Luna.
Il sensore è un dispositivo CCD con un selettore di filtro a sei posizioni:

- un filtro a banda larga da 400 nm a 800 nm;
- quattro filtri centrati rispettivamente in 415 nm (con larghezza di banda di 40 nm), 560 nm (10 nm), 650 nm (10 nm), 750 nm (20 nm);
- un filtro opaco di protezione per l'intesificatore di immagine.

L'esperimento è stato condotto con il solo scopo di acquisire delle immagini dettagliate delle zone polari e di alcuni luoghi di particolare importanza come i siti di atterraggio di precedenti missioni. Quest'ultime hanno il pregio di essere state acquisite con più bande (e quindi risultare a colori), mentre per le prime si è utilizzata la sola banda a 750nm. Nominalmente la risoluzione varia intorno ai 10 metri/pixel, anche se a causa delle imprecisioni dovute alla tecnologia si arriva ad un massimo di 40 metri/pixel.

La camera NIR

La camera NIR monta una lente catadiottrica e permette l'utilizzo di 6 bande elettromagnetiche: 1100 nm (larghezza di banda di 30 nm), 1250 nm (30 nm), 1500 nm (30 nm), 2000 nm (30 nm), 2600 nm (30 nm), 2690 nm (60 nm).
L'obiettivo principale dell'esperimento era di assecondare le ricerche per la mappatura mineralogica della Luna. A tale scopo, integrando tali dati con quelli della camera UVVIS, si copre il fabbisogno di informazioni con i dati di ben 11 bande elettromagnetiche disposte nell'intervallo da 415 nm a 2690 nm. Anche qui, come per UVVIS, la risoluzione varia dai 100 ai 400 metri/pixel.

Figura 2.3. La camera HIRES e la camera NIR

La camera LWIR

Progettualmente simile alla camera NIR, anche la LWIR utilizza una lente catadiottrica, ma adotta un unico filtro con banda centrale a 8750 nm con larghezza di 1500 nm.
In base anche alla caratteristica sopra descritta, si intuisce un obiettivo differente, che consiste in osservazioni delle condizioni termiche della superficie.

Figura 2.4. La camera LWIR e una della due Star Tracker

Le camere Star Tracker A e Star Tracker B

La due camere Star Tracker sono identiche in tutto e per tutto. La loro duplice presenza è richiesta per motivi di ridondanza e quindi di sicurezza di acquisizione dei dati. Anche per questo motivo, sono montate su lati opposti della navetta, in modo che eventuali interferenze solari estemporanee pregiudichino il funzionamento di solo una delle due.
La funzione primaria è stata quella di calcolare dei riferimenti angolari assoluti, basandosi sul raffronto tra l'intero panorama stellare da esse percepito e l'omologa mappa di stelle caricata nel sistema di bordo prima del lancio, per poter determinare l'assetto e, quindi, il sistema di riferimento spaziale. Secondariamente, gli obiettivi prettamente scientifici si sono limitati alle osservazioni di particolari eventi riguardanti Terra, Luna e Sole, come, ad esempio, le immagini della superficie lunare illuminata dalla luce della Terra.
Le due camere hanno una struttura ad ottiche concentriche con un campo di fibra ottica appiattito per associare l'immagine al CCD.

Strategia di acquisizione

Durante il corso dei 71 giorni in orbita, Clementine ha mappato sistematicamente 38 milioni di chilometri quadrati di superficie lunare e, come abbiamo visto, a differenti lunghezze d'onda, dai 415 nm dell'ultravioletto fino ai 2800 nm dell'infrarosso, per un totale di circa un milione di immagini. In aggiunta, sono state acquisite 620 mila immagini termiche ad alta risoluzione e 320 mila a medio-infrarosso, è stata mappata la topografia della Luna con equipaggiamento laser ed ampliata la conoscenza per quel che concerne la gravità. Tutti i dati così ottenuti, sono serviti per chiarire i dubbi e migliorare le conoscenze su vari aspetti dell'ambiente lunare, come la possibilità di trovare ghiaccio al polo sud, la composizione della crosta lunare e dei basalti, financo la natura geologica del sito di atterraggio della missione *Apollo 16*.
La strategia di acquisizione durante la mappatura sistematica è stata limitata principalmente dalla quantità di dati che potevano essere scaricati: solitamente si sono raggiunti i 100 MB per orbita, fatti salvi i periodi di blocco delle radiofrequenze in cui si arrivava a 60 MB. Il periodo di blocco scattava allorquando la navetta veniva a trovarsi nascosta, rispetto alla Terra, dalla Luna.
Nonostante tali restrizioni, le decisioni hanno dovuto tenere conto soprattutto delle finalità a cui si voleva giungere:

- copertura globale con 5 bande UVVIS e 6 NIR;
- acquisizione continua con LWIR;
- immagini delle zone polari con HIRES;
- immagini HIRES addizionali;
- avere due collezioni di dati UVVIS, una con tempo di esposizione grande e l'altro piccolo, in modo da ovviare a problemi di saturazione di colori.

Sostanzialmente, le osservazioni della Luna si possono suddividere in quattro fasi:

1. Orbite 1-31. Periodo di prova durante il quale i metodi di acquisizione delle informazioni sono stati testati e rifiniti, se necessario. Particolari osservazioni, come i luoghi di atterraggio delle missioni antecedenti, sono avvenute in tale periodo, durante il quale la navetta è rimasta in orbita terrestre.

2. Orbite 32-168. Ingresso nell'orbita lunare e primo mese di mappatura sistematica della superficie lunare: periselene (punto, dell'orbita lunare, più vicino al centro di massa) a 30° latitudine Sud. Data di inzio: 26 Febbraio 1994. Data di fine: 26 Marzo 1994.

3. Orbite 169-297. Secondo mese di mappatura sistematica della superficie lunare e successiva uscita dall'orbita: periselene a 30° latitudine Nord. Data di inizio: 26 Marzo 1994. Data di fine: 21 Aprile 1994.

4. Orbite 298-348. Periodo conclusivo riservato a colmare eventuali mancanze nella collezione di informazioni, acquisire i dati di obiettivi particolari, calibrare i dati ottenuti. Anche in tale periodo la navetta è rimasta in orbita terrestre.

Le orbite che contraddistinguono il secondo ed il terzo periodo consistono in una fase di mappatura lunare di 80 minuti vicino al periselene e 139 minuti di invio dati vicino all'aposelene (punto, dell'orbita lunare, più lontano dal centro di massa).
L'intera strategia, per come è stata concepita, ha permesso la copertura globale, in termini di immagini ed altimetrie, da 60°S a 60°N, per un totale di 300 orbite.
Ci si potrebbe dilungare a dismisura, focalizzando l'attenzione sulle tecniche utilizzate per ogni singola tecnologia, nonché addentrandosi nelle precise caratteristiche delle apparecchiature descritte, ma, oltre ad esulare dagli scopi tesistici, risulterebbe solamente un'arida dimostrazione numerica di come i tecnici e gli ingegneri della NASA abbiano saputo e voluto combinare tutte le informazioni a loro disposizione nella speranza, non sempre verificata, di ottenere dati e, soprattutto, immagini il più scientificamente corrette.

2.3 Il formato dei dati

La presenza dei dati NASA ha permesso la realizzazione dell'intero progetto di tesi in discussione. L'enorme quantità di informazioni reperibili per ogni singola missione spaziale, peraltro, ha reso necessario istituire una serie di regole per la categorizzazione di tutti i dati a partire dai più semplici, come testi ed immagini, fino ad arrivare alle quote altimetriche o alle rilevazioni spettrografiche: l'insieme delle indicazioni per organizzarli in modo omogeneo e di pubblica utilità è racchiuso nello standard PDS, compiutamente esposto in [9].
Una volta creato l'enorme collezione di informazioni si è pensato di creare anche un'insieme di metodi per poterli gestire, manipolare e modificare. Tale raccolta altro non è che una libreria di funzioni compilabili in alcuni linguaggi tra cui il C/C++.

2.3.1 Lo standard PDS

Il PDS (Planetary Data System) è lo standard con cui la NASA acquisisce, archivia e distribuisce gran parte dei dati collezionati riguardo le entità del nostro sistema solare. Fin dalla sua prima realizzazione prototipale del 1982, era chiara la totale assenza di standard per i dati prodotti dagli scienziati sulla base delle missioni spaziali. Così il PDS avrebbe dovuto essere in grado di gestire i data-set (archivi di dati) in variegati formati.
Gli sviluppatori del prototipo di PDS risposero a tali esigenze in vari modi. In primis venne concepito un semplice linguaggio con keyword (parole-chiave indicanti gli attributi delle entità da gestire) in modo che i data-set e i singoli file potessero essere etichettati identificandone formati e contenuti: tale linguaggio sfociò nel finale ODL (Object Description Language). Oggi le etichette PDS scritte in ODL sono associate ad (ma alle volte anche inglobate in) ogni file che fluisce nel PDS, sia in ingresso che in uscita. È importante capire che una generica etichetta può essere letta oltre che dai computer, anche dalle persone. Per esempio, uno scienziato può leggere l'etichetta PDS per capire di che tipo

è il file in suo possesso e, contestualmente, il software utilizzato per l'analisi dei dati può analizzare l'etichetta e usare le informazioni ivi contenute per identificare, localizzare e cercare i dati nel file.

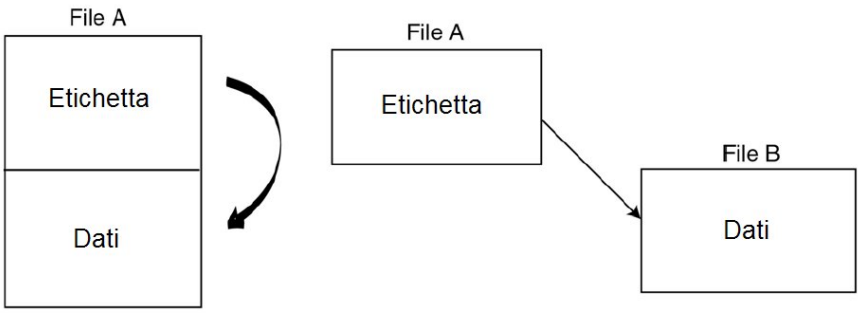

Figura 2.5. Metodi di archiviazione dei file secondo PDS-ISIS

Il secondo passo per la realizzazione dello standard ha portato la comunità astronomica e scientifica a sviluppare PSDD (Planetary Science Data Dictionary) e definire formalmente, così, i tipi standard degli oggetti di dati, come immagini, spettri e tabelle, utilizzati dagli studiosi. PSDD, oltre a quelle degli oggetti standard PDS, contiene le definizioni dei termini elementari usati per la descrizione dei dati astronomici, come latitudine e longitudine. Pertanto, la definizione di un oggetto specifica un set di termini elementari che rappresentano gli attributi di quello specifico oggetto. Un attributo, infine, descrive e qualifica un aspetto del formato o del contenuto dell'oggetto stesso: alcuni di essi sono obbligatori (ad ogni istanziazione deve essere assegnato un valore), altri opzionali.
Gli applicativi che gestiscono e analizzano i dati astronomici spesso richiedono l'accesso a combinazioni di oggetti standard e ad informazioni contenute sia nel PSDD sia nelle etichette PDS. Eccone, di seguito, due esempi.

- Se uno scienziato fornisce un file al sistema PDS, deve esserci associata obbligatoriamente un'etichetta. In tal caso le informazioni del PSDD vengono usate per assicurare la correttezza sintattica delle etichette correlate ai dati.

- Quando, invece, PDS produce un file, le informazioni del PSDD possono essere usate per creare l'etichetta PDS associata ai dati. I valori degli attributi obbligatori dell'oggetto vengono inizializzati per identificare l'instanziazione dello specifico oggetto. Gli attributi e i valori ad essi associati sono quindi codificati in ODL per creare l'etichetta.

Sebbene questo sistema di etichette renda possibile la totale gestione degli oggetti, sfortunatamente c'è carenza di software ad ampio spettro per l'analisi e la gestione di dati PDS. Non esistono direttive condivise per manipolare gli oggetti PDS, così che un applicativo che ne gestisce un tipo potrebbe funzionare in modo assai differente da un altro che opera su tipi diversi. In più, alcuni oggetti standard sono supportati da parecchi software,

mentre altri ne hanno davvero pochi a disposizione. Esistono programmi che addirittura gestiscono solo un sotto-gruppo di un certo tipo di dati riuscendo a manipolare, ad esempio, oggetti di tipo immagine a singola banda, ma non multi-banda. Infine, altri ancora, riescono a gestire solamente oggetti formattati per particolari macchine o, comunque, codificati in file in modi specifici. L'intento di OAL è risolvere alcuni dei problemi ora citati per mezzo di direttive ad ampia condivisione e diffusione.

Struttura e scopi del PSDD

L'obiettivo primario del PSDD è di mettere in condizione i membri della comunità scientifica planetaria di beneficiare degli immani sforzi compiuti nell'uniformare e regolarizzare il gergo scientifico, trasformandolo in un linguaggio vero e proprio. Questo lo porta ad assurgere un ruolo fondamentale come guida verso sistemi di dati ancora in sviluppo, ma anche per tutti quei sistemi che in qualche maniera si riconducono allo standard PDS o a quello AMMOS (Advanced Multimission Operations System) usato da SFOC (Space Flight Operations Center). Secondariamente, PSDD serve come punto di contatto tra PDS e MGDS (Multimission Ground Data System), quest'ultimo sviluppato da AMMOS. Il PSDD altro non è che un dizionario composto di tre parti:

- le regole seguite per la nomenclatura e la descrizione degli elementi, che possono essere semplici definizioni di termini, ma anche specifici attributi utilizzati nel linguaggio ODL;
- una lista alfabeticamente ordinata di tali elementi, con le precisazioni tecniche del caso;
- una serie di appendici in cui viene spiegato l'utilizzo degli elementi stessi.

Per maggiori e più dettagliate informazioni si faccia riferimento a [10].

L'oggetto di dati PDS

Un oggetto PDS è semplicemente un insieme di dati di tipo riconoscibile e può essere un banale numero intero ma anche un'immagine multi-spettro. Inoltre è definito nel PSDD e per ognuno di essi esiste (o può essere creata in caso di mancanza) un'etichetta PDS che fornisce i valori degli attributi dell'oggetto in questione.
L'uso della parola oggetto merita attenzione. Lo standard PDS è basato sugli oggetti ma non ad essi orientato. Il basarsi sugli oggetti vuole significare che PDS definisce un set di tipi di dati adatto a descrivere una larga varietà di oggetti che gli studiosi conoscono e manipolano ordinariamente come immagini, tabelle e altro. Per ogni tipo di dato, nel PSDD è definito un gruppo di attributi che descrive i dati in modo che possano essere compresi sia dalle persone sia dai computer. In una certa misura, gli oggetti PDS sono simili ai tipi di dati di un sistema orientato agli oggetti, fatto salvo per poche ma essenziali caratteristiche mancanti:

- ereditarietà: meccanismo di relazione esplicita tra un tipo di oggetto ed altri oggetti da esso derivati;

- metodi: meccanismi di codice associato a specifici tipi di oggetti.

Un oggetto PDS è composto dall'etichetta e dai dati.

Etichetta : contiene la descrizione dell'oggetto con i valori degli attributi per quel dato oggetto. Tale informazione può essere in formato esterno (in ODL dentro un file) o interno (in alcune rappresentazioni residenti in memoria). La rappresentazione standard di un'etichetta PDS residente in memoria è un albero ODL. Un albero ODL è una struttura dati interconnessa che contiene le medesime informazioni dell'etichetta PDS corrispondente, ma in un formato di più facile accesso e gestione per l'elaboratore elettronico. Per le operazioni di gestione e di trasformazione da etichetta PDS ad albero ODL, esiste la libreria L3 (Label Library Lite), inclusa nella distribuzione di OAL.

Dati : sequenza di byte che forma il valore primario dell'oggetto. Il formato è descritto dall'etichetta PDS associata. Questa parte dell'oggetto PDS può anche essere indefinita in modo da non avere altro se non i valori degli attributi. Non è, però, consentito l'opposto: una collezione di dati senza etichetta non può essere considerato un oggetto PDS.

2.3.2 Object Access Library

La libreria OAL (Object Access Library) è una serie di routine software che si integra con gli standard definiti in PDS. Il suo intento è proporre linee guida e direttive ad ampia scala per la realizzazione di applicativi per la gestione di oggetti PDS. Il tutto tramite una consistente collezione di funzioni, per la lettura e la scrittura di oggetti PDS da e in una vasta gamma di formati di file, che permettono di tradurre i dati nella rappresentazione più appropriata per un dato computer. Sono previste anche le routine di manipolazione per la maggior parte dei tipi di oggetti PDS. OAL, quindi, è un insieme di metodi base per realizzare, in modo semplice, applicativi congrui alle esigenze di PDS.
OAL utilizza la libreria di funzioni L3 e si compone di tre livelli operativi: *Stream*, *Struttura*, *Oggetto*.
Le documentazioni di OAL [11] e di L3 [12] sono ambo distribuite assieme all'intero pacchetto software. Ci si rinvia a queste per ulteriori e dettegliate spiegazioni.

Livello Stream

Il livello Stream fornisce le funzioni necessarie per leggere/scrivere da/su file un oggetto PDS o parte di esso. Consente all'applicazione di operare sui dati come se fossero un flusso di byte contigui. Permette, inoltre, di estrarre gruppi di dati da semplici file di lunghezza e fissa e variabile. Il livello Stream è indipendente così da poter essere usato senza tutta la libreria OAL.

Livello Struttura

Il livello Struttura affronta i problemi di rappresentazione dei dati, come la conversione numerica dei dati in formati compatibili a seconda della piattaforma utilizzata. In definitiva, consente all'applicazione di occuparsi dei dati come fossero una sequenza di tipi di dati atomici come numeri interi o caratteri ASCII.
Come esempio, si consideri una semplice tabella di dati binari dove ogni riga consiste in una serie di campi, ognuno dei quali è un valore atomico. Il livello Struttura consente la rappresentazione di questi dati su diversi tipi di piattaforme; è in grado di convertire, infatti, i campi della tabella, ad esempio, dal formato VAX a quello Sun.
L'altro ruolo importante, il livello lo svolge nell'allineamento dei dati in memoria, visto che, a seconda dei sistemi informatici e prendendo in esame ancora la nostra tabella d'esempio, possono essere allineati su singoli byte, su word (insieme di byte) o su multi-word.

Livello Oggetto

Il livello Oggetto fornisce i metodi per gli utenti finali risultando, così, l'interfaccia primaria tra il singolo studioso e l'intero mondo PDS. Comprende, pertanto, le funzioni di base per leggere, manipolare e scrivere oggetti PDS, ma è anche possibile creare delle proprie routine da condividere con la comunità scientifica.

Libreria L3

La libreria L3 viene utilizzata da tutti i livelli OAL sopra menzionati. Supporta le funzioni per controllare la correttezza sintattica delle etichette PDS e per accedervi in memoria. In generale, gran parte delle routine di tale libreria, servono per lo scambio di informazioni tra l'etichetta PDS e il suo omologo albero ODL residente in memoria, ma anche per la gestione di quest'ultimo.

Capitolo 3

Importatore delle immagini

3.1 Panoramica

L'importatore delle immagini è il primo dei due programmi aggiunti ad UltraTools [cfr. paragrafo 1.1.5]. Il suo ruolo è quello di creare una libreria di immagini della superficie lunare che servirà da punto di partenza per le successive elaborazioni dell'adattatore [cfr. capitolo 6], descritto in seguito. Si è deciso che tale collezione dovesse essere formata da file di tipo 'bmp' aderenti allo standard RGB in modo da conformarsi alle esigenze del sistema UltraPort [cfr. paragrafo 1.1.3].
Per avere un'idea dei passaggi per arrivare ad avere il database del PID si può fare riferimento alla figura 3.1 che cerca, in maniera intuitiva, di rendere chiara la trafila, a grandi linee, delle operazioni da eseguire.

Figura 3.1. Trafila per la realizzazione del database del PID

Per giungere all'obiettivo ora esposto, si è dovuto studiare il formato delle immagini della NASA, messe a disposizione tramite il proprio sito internet [2], aderenti allo standard PDS [cfr. paragrafo 2.3.1]. Alla fine si è deciso di utilizzare i dati della missione Clementine [cfr. sezione 2.2] ed in particolare del progetto denominato DIM (Digital Image Model) creato con le immagini ricavate dalle osservazioni spettrometriche della camera UVVIS [cfr. paragrafo 2.2.2] e memorizzate in file denominati "oggetti-cubo".
Il programma, realizzato in C/C++, è composto da più classi [cfr. capitolo 5] che interagiscono tra loro e che basano alcune proprie operazioni sulla libreria di funzioni OAL [cfr. paragrafo 2.3.2], creata appositamente dalla NASA per venire incontro agli sviluppatori per diffondere e migliorare tale sistema di condivisione di dati scientifici.
In questo capitolo ci si addentrerà non solo nelle funzionalità delle singole classi da me

scritte, ma anche in alcune delle routine OAL che sono servite per la realizzazione dell'archivio di immagini.
Si suppone che il lettore abbia almeno le conoscenze di base per capire il linguaggio C/C++ e i concetti elementari della grafica digitale.

3.2 Il modello di colori RGB

Nel 1931 la CIE (Commission Internationale dell'Enclairage) ha descritto le specifiche del modello di colori RGB, per la codifica di immagini digitali. Questo è di tipo additivo e si basa su tre colori: rosso (Red), verde (Green) e blu (Blue). Una qualsiasi immagine, attraverso filtri o particolari tecniche, può essere scomposta in questi colori di base che, miscelati opportunamente tra loro, rendono quasi tutto lo spettro dei colori visibili, ad eccezione delle porpore. Nello specifico campo fisico, ad ognuno dei tre colori principali, corrisponde una radiazione luminosa: 700 nm per il rosso, 546,1 nm per il verde, 435,8 nm per il blu.
Tale standard, come detto, è di tipo addittivo. Ciò vuol dire che unendo le tre componenti alla loro massima intensità si ottiene il bianco. Combinando i valori a coppie si ottengono il ciano, il magenta ed il giallo, come mostrato in figura 3.2.

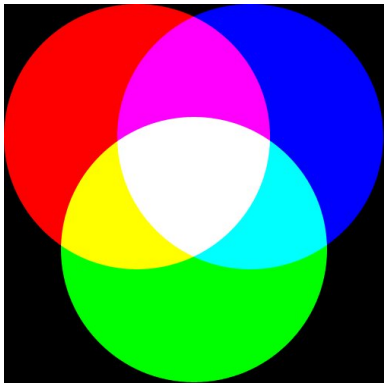

Figura 3.2. Esempio dei colori RGB

3.3 Il progetto DIM

Il DIM (Digital Image Model) è un progetto realizzato dalla USGS (United States Geological Survey) sotto la direzione di Eric M. Eliason (referente USGS) e Alfred McEwen (Università dell'Arizona) in stretta collaborazione con la NASA.
Nel suo insieme, è un mosaico composto da oltre quattrocentomila immagini, contenute nell'archivio EDR (Engineering Data Record) della missione Clementine, ottenute dalle rilevazioni multi-spettro della camera UVVIS montata sulla navicella, e aderenti alle direttive PDS. L'intero mosaico è mappato secondo la proiezione cartografica SEA (Sinusoidal

Equal-Area) alla risoluzione di 100 metri per pixel e nella sua interezza richiede un ammontare di spazio fisico di circa 66 gigabyte, che alla NASA hanno pensato di suddividere in 78 volumi, in modo che ognuno possa essere archiviato in un normale cd.
L'intero mosaico UVVIS è stato generato utilizzando il sistema di elaborazione ISIS (Integrated System for Imagers and Spectrometers), il quale ha permesso complesse operazioni di correzione e normalizzazione dei dati grezzi, nonché l'effettiva realizzazione di questa sorta di mastodontico puzzle.
Questo database è partizionato in quadranti (o tessere) come chiaramente visibile in figura 3.3, in cui le linee in neretto indicano le porzioni di superficie coperte da ognuno dei 78 volumi, identificati univocamente dai numeri in grigio. All'interno dei settori 1, 2, 3, 4, 5, 6, 7, 8, 9, 76, 77, 78 della stessa figura si può osservare la suddivisione in tessere, ognuna delle quali corrisponde ad un file immagine 'img'.

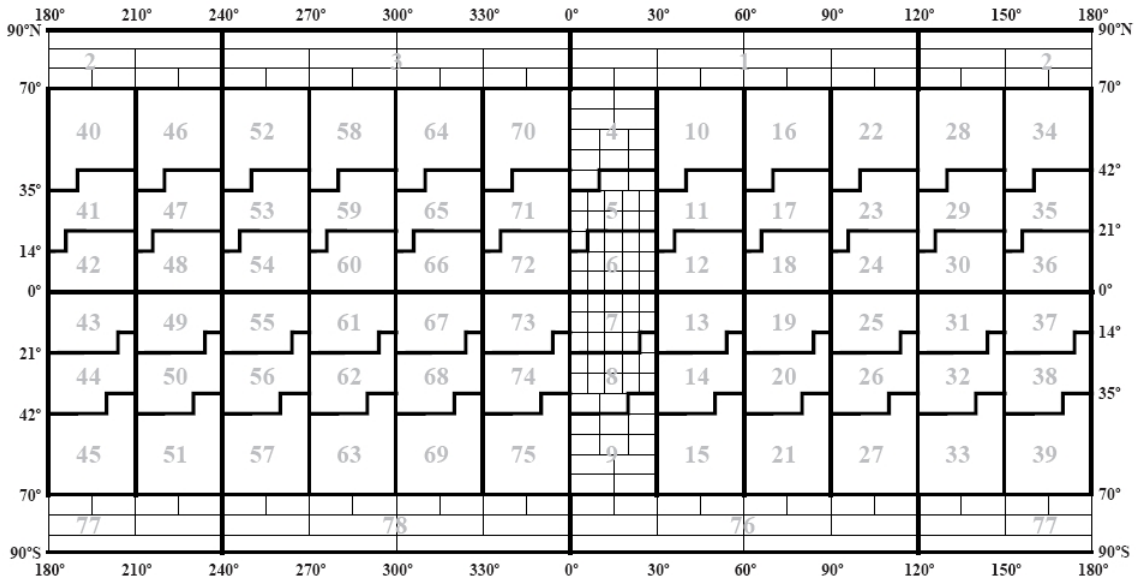

Figura 3.3. La proiezione cartografica Sinusoidal Equal-Area

Il contenuto di ogni singolo volume comprende, oltre ai file immagine dei quadranti, anche informazioni e dati addizionali, come, ad esempio, un database di immagini in formato 'jpg' in tre diverse dimensioni e, inevitabilmente, con risoluzione assai minore rispetto alle immagini UVVIS di tipo 'img', a partire da 500 metri fino a 12,5 kilometri per pixel.
Per ogni volume è previsto, inoltre, un esploratore digitale (una sorta di navigatore che funziona utilizzando il browser internet) con cui è possibile avere una visuale globale e particolare delle immagini contenute nel medesimo, con la sola limitazione di accontentarsi di una bassa risoluzione. L'intera documentazione, infine, è completata da file che descrivono gli aspetti della missione Clementine, della navetta e degli strumenti di bordo. Tutti i file (immagini, tabelle, etichette o quant'altro) sono codificati come sequenze di caratteri ASCII e composti da righe separate l'un l'altra dalla combinazione di caratteri

< cr > < lf > (carriage-return e line-feed), in modo che i vari sistemi informatici riescano a riconoscere la fine della riga.

3.4 Gli oggetti-cubo

Gli oggetti-cubo sono in realtà dei file in grado di essere elaborati dai comuni programmi ISIS e, codificati in un formato simile al PDS, si compongono di due parti: un etichetta, generalmente localizzata all'inizio del file, e l'area dei dati, che può contenere uno o più sotto-oggetti.
I cubi sono il caso tridimensionale dei "qubi", anch'essi supportati da ISIS, che hanno la caratteristica di poter memorizzare i dati in forma di vettori a più dimensioni, con un limite massimo di 6.
Ogni cubo viene normalmente utilizzato per archiviare i dati prodotti dagli spettrometri per immagini. Non a caso, hanno tre dimensioni, in quanto due corrispondono ai canonici assi spaziali bidimensionali, mentre la terza rappresenta la coordinata spettrale.
Anche se i cubi possono contenere innumerevoli oggetti, obbligatoriamente ve ne si trovano due:

- HISTORY è una sorta di registro che memorizza le operazioni che vengono svolte sull'oggetto-cubo e che portano a modificarlo e quindi aggiornarlo;

- QUBE contiene le informazioni basilari per il riconoscimento e la localizzazione dei dati veri e propri.

Per qualsiasi chiarimento più particolareggiato, si consiglia di prendere visione del sito internet di riferimento ISIS [13].

3.4.1 Etichetta ISIS

L'etichetta è solitamente localizzata all'inizio del file contenente i dati, ma può anche essere separata dai dati e presentarsi in un file indipendente. In tal caso, è prassi comune dotarlo di un'estensione 'lbl' o 'lab' (dal vocabolo inglese "label") e al suo interno prevede un puntatore al file contenente i dati. Ad ogni modo, descrive la struttura ed il contenuto del file, utilizzando una codifica abbastanza semplice nella forma di "*attributo* = *valore*".
Visivamente si può fare riferimento alla figura 2.5.
All'interno della parte dell'etichetta che riguarda l'oggetto QUBE è possibile trovare una più precisa descrizione dei dati tramite gli specifici valori degli attributi, alle volte anche riuniti in sotto-gruppi come, ad esempio, quello definito con GROUP = BAND_BIN che specifica le caratteristiche delle bande elettromagnetiche usate per l'acquisizione dei valori.
Alcuni sotto-gruppi possono fare riferimento ai dati essenziali del cubo, altri possono essere a supporto di quest'ultimi, altri ancora possono dare indicazione di generiche informazioni che rivestono un ruolo non prioritario ma comunque importante.

Esempio di etichetta

```
/* File Structure */

RECORD_TYPE = FIXED_LENGTH
RECORD_BYTES = 512
FILE_RECORDS = 64958

^HISTORY = $ISISDATA/null_history.dat
OBJECT = HISTORY
END_OBJECT = HISTORY

^QUBE = (ui73n007.img,6249<BYTES>)
OBJECT = QUBE
/* Qube object description */
/* Qube structure */
AXES = 3
AXIS_NAME = (SAMPLE,LINE,BAND)

/* Core description */
CORE_ITEMS = (1562,2127,5)
CORE_ITEM_BYTES = 2
CORE_ITEM_TYPE = SUN_INTEGER
(...)

/* Suffix description */
SUFFIX_BYTES = 4
SUFFIX_ITEMS = (0,0,0)
CORE_NAME = RAW_DATA_NUMBER
CORE_UNIT = NONE
(...)

GROUP = BAND_BIN
BAND_BIN_ORIGINAL_BAND = (1,2,3,4,5)
BAND_BIN_UNIT = MICROMETER
BAND_BIN_FILTER_NAME = (A,B,C,D,E)
BAND_BIN_CENTER = (0.415000,0.750000,0.900000,0.950000,1.000000)
BAND_BIN_WIDTH = (0.040000,0.010000,0.020000,0.030000,0.030000)
END_GROUP = BAND_BIN

GROUP = IMAGE_MAP_PROJECTION
A_AXIS_RADIUS = 1737.4000000
(...)
EASTERNMOST_LONGITUDE = 15.0481997
(...)
END_GROUP = IMAGE_MAP_PROJECTION
END_OBJECT = QUBE
END
```

3.4.2 Formato dei cubi

Innanzitutto è importante chiarire che i cubi hanno un formato fisico in cui vengono effettivamente memorizzati. Tuttavia, per comprenderne la struttura, è bene avere in mente

anche la rappresentazione logica o concettuale, che ne facilita l'intellegibilità, mostrata in figura 3.4.
Un oggetto-cubo presenta, sostanzialmente, un'area contenente i dati primari e varie, ma non sempre necessarie, di complemento, per meglio strutturare alcuni dati primari, oltre che per dare informazioni generiche.

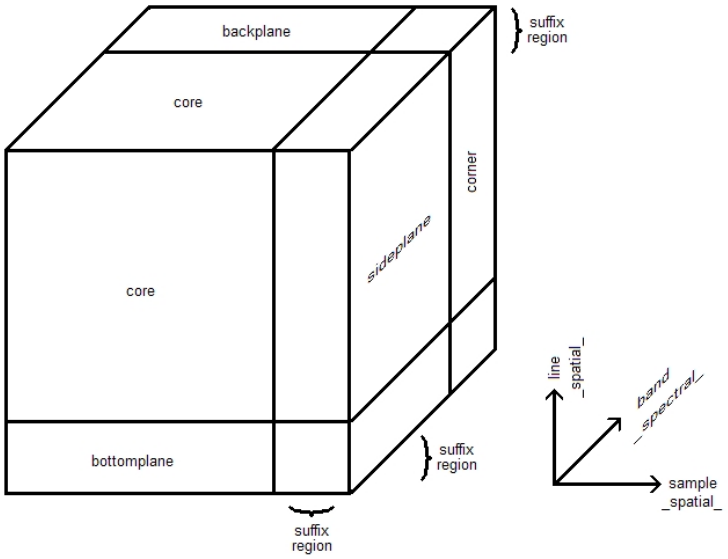

Figura 3.4. Rappresentazione logica di un oggetto-cubo

Area primaria

Il nucleo, o regione *CORE*, è l'area di fondamentale importanza in cui è presente la descrizione principale dei dati memorizzati in un file cubo. Solitamente, per un cubo aderente allo standard ISIS, tale regione corrisponde ai dati che descrivono spettralmente le immagini, ovvero consiste in un certo numero di immagini piane ognuna referente ad una precisa banda e, quindi, ad una diversa lunghezza d'onda.
Nell'etichetta si riconoscono importanti attributi riferiti a tale regione, quali:

- AXES: numero degli assi nel nucleo, che per un cubo è sempre pari a 3;
- AXIS_NAME: nome degli assi nell'ordine in cui sono stati memorizzati, che per un cubo solitamente è la terna (SAMPLE, LINE, BAND);
- CORE_ITEMS: lunghezza in pixel di ognuno degli assi principali;
- CORE_ITEM_BYTES: dimensione in byte occupata da un singolo valore;
- CORE_ITEM_TYPE: tipo della rappresentazione interna dei valori memorizzati;

- CORE_NAME: nome per i valori memorizzati;
- CORE_UNIT: unità di misura dei dati.

Piani di complemento

Ognuno dei tre assi principali può, opzionalmente, avere dei dati di supporto che estendono la lunghezza degli assi stessi. Concettualmente, queste estensioni possono essere viste come piani che vengono attaccati al nucleo.
Tali regioni, denominate piani *SUFFIX*, possono integrare il nucleo in una qualsiasi delle tre dimensioni, anche in tutte contemporaneamente, a seconda della presenza o meno di dati supplementari che riguardino l'asse principale che vanno ad estendere.
Inoltre, in base all'asse primario integrato, acquisiscono nomi differenti:

- BACKPLANE per l'asse BAND;
- SIDEPLANE per l'asse SAMPLE;
- BOTTOMPLANE per l'asse LINE.

Giusto per capire, un BACKPLANE potrebbe essere usato per immagazzinare i valori di latitudine di ogni pixel. Un altro BACKPLANE potrebbe contenere, pixel per pixel, la lunghezza d'onda con maggior assorbimento dell'intero spettro a disposizione. Uno o più SIDEPLANE potrebbero venire utilizzati per dati ingegneristici associati ad ogni linea di dati.
Nell'etichetta si riconoscono importanti attributi riferiti a tale regione, quali:

- SUFFIX_BYTES: dimensione in byte dei dati complementari;
- SUFFIX_ITEMS: numero dei piani di complemento, se presenti, riferiti agli assi principali nell'ordine in cui sono stati dichiarati in AXIS_NAME;
- *axis*_SUFFIX_NAME: nome del piano di complemento referente all'asse primario *axis*;
- *axis*_SUFFIX_UNIT: unità di misura dei dati del piano di complemento referente all'asse primario *axis*.

Aree d'angolo

Se un oggetto-cubo include piani SUFFIX su più di un asse primario, inevitabilmente si vengono a formare delle intersezioni tra due di questi (o anche tutti e tre) che danno luogo alle cosiddette regioni *CORNER*. Lo spazio in memoria per tali regioni viene allocato, sebbene allo stato attuale non vengano utilizzate da nessun applicativo.

3.5 Le immagini NASA

I file che abbiamo utilizzato per la creazione del nostro database di immagini lunari appartengono all'enorme collezione di ortofoto acquisite grazie alla missione Clementine. Questi file grafici con estensione 'img' sono associati ognuno alla propria etichetta 'lab', il cui formato è stato precedentemente descritto [cfr. paragrafo 3.4.1]. D'ora in avanti, quando si farà un generico riferimento ad un file, si considererà la coppia, salvo diversamente ed esplicitamente dichiarato. Inoltre, proprio per la particolare struttura, ogni coppia deve essere archiviata nella stessa cartella, altrimenti i riferimenti tra file non funzionerebbero più nella maniera auspicata.
Lo standard utilizzato per la memorizzazione e l'archiviazione dei dati è una commistione tra PDS e ISIS, e quindi, se non diversamente esplicato, quando verrà nominato uno ci si riferirà all'insieme dei due.

3.5.1 Struttura del file immagine

Ogni file immagine contiene un'intestazione (o etichetta) all'inizio del file, seguita dai dati immagine. Il numero dei byte dell'intestazione è un multiplo del numero di byte che compongono una linea dell'immagine secondo la formula $numero_{campioni} \times 2$ byte/pixel. L'intestazione contiene i dati che descrivono il file immagine in forma di testo ASCII e può, perciò, essere aperta con un semplice editor di testi.
Il mosaico UVVIS è composto di file immagine i cui pixel sono codificati come interi segnati su 16 bit. I pixel sono ordinati secondo le regola MSB (Most Significant Byte), ovvero il primo byte è quello più significativo. MSB è il metodo utilizzato dai sistemi Unix/Sun e Macintosh. Per tutti gli altri (IBM compatibili, Dec/Alpha, VAX) è necessario scambiare l'ordine dei byte prima di poter utilizzare i dati.

3.5.2 Definizione geometrica di un pixel

Lo scopo di questo paragrafo è meno ovvio di quanto non possa sembrare e consiste nel fugare i dubbi sul concetto di pixel e spiegare le convenzioni utilizzate dalla NASA a riguardo.

Concezione puntuale e spaziale

La definizione di punto geometrico potrebbe convenire quando si lavora con griglie di coordinate, con il risultato di avere un numero dispari di pixel per una mappa con un numero pari di incrementi spaziali. Per cambiamenti di scala, però, potrebbero sorgere i primi limiti di tale definizione. Trattare il pixel come area, al contrario, consentirebbe di avere un numero pari sia di pixel che di incrementi spaziali e renderebbe il cambiamento di scala un semplice formalità. Ma ora, la griglia, matematicamente parlando, cadrebbe in mezzo ai pixel.
Si è deciso che il concetto di area fosse la scelta migliore, sebbene sia necessario risolvere

alcuni problemi di asimmetrie che inevitabilmente si introducono nelle cartografie. Le soluzioni non mancano:

1. usare, per la griglia di coordinate, linee dello spessore di 2 pixel;
2. spezzare i pixel a cavallo di una linea secondo calcoli matematici;
3. adottare una convenzione per la quale le linee della griglia vengono sistematicamente disegnate sfalsate rispetto alla loro reale posizione matematica.

Conversione tra coordinate intere e reali

I pixel sono identificati da una coppia di numeri (x,y) che indica l'indice del campione, ovvero della colonna, e quello della linea. La convenzione per la numerazione prevede che il primo pixel sia quello nell'angolo in alto a sinistra dell'immagine con coordinate (1,1) e che l'incremento sulle righe avvenga da sinistra verso destra, mentre quello sulle colonne dall'alto verso il basso.
A questo punto esistono tre diverse possibilità per allineare un sistema di coordinate reali a questa scacchiera di pixel (in gergo chiamata *mesh*): stabilire se il punto di coordinate (1.0,1.0) sia l'angolo in alto a sinistra, il centro, o l'angolo in basso a destra del pixel (1,1). Convenzioni storiche hanno portato a scegliere che le coordinate reali del punto corrispondano al centro del pixel.
In altri termini:

- il punto (1.0,1.0) è collocato al centro del pixel (1,1);
- l'angolo in alto a sinistra ha coordinate (0.5,0.5);
- l'angolo in basso a destra ha coordinate $(1.4\overline{9},1.4\overline{9})$;

Convensione cartografica

La rappresentazione cartografica di un pixel è, in termini matematici, aperta sui limiti di incremento (in basso e a destra), e chiusa altrimenti. L'unica eccezione è prevista per i confini fisici della proiezione cartografica, e quindi il limite inferiore dei pixel più in basso è chiuso ed include il bordo della proiezione stessa.

Linee della griglia

Ultimo passo, è necessario scegliere delle regole per disegnare la griglia delle coordinate sulle immagini planetarie o sulle intere mappe. La convenzione utilizzata è quella di disegnare le linee in quei pixel che ne contengono il valore reale prima che venga raggiunto il limite aperto. L'unica eccezione riguarda i valori di latitudine e longitudine sui bordi, le cui linee devono sempre apparire.
Per capire e riferendosi alla figura 3.5, una griglia a 10° di ampiezza parte dal pixel 1 e viene disegnata ogni 10 pixel (1, 11, 21, 31, ...) fino al raggiungimento del bordo. L'ultima linea verrà così tracciata sul pixel che precede proprio tale limite (riga 180 invece di 181, o colonna 360 anziché 361).

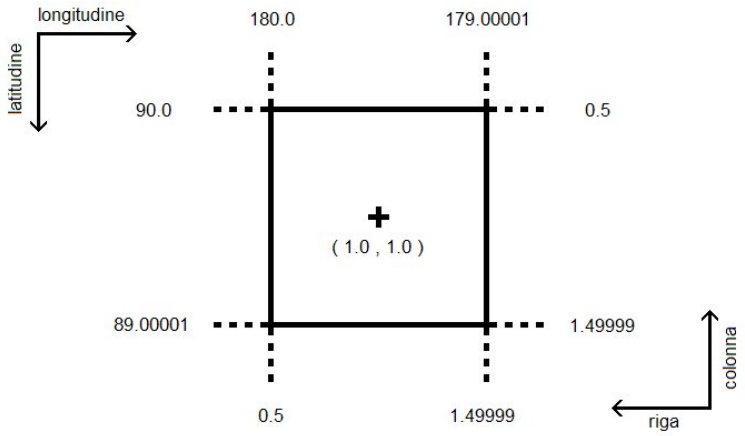

Figura 3.5. Esempio del pixel (1,1)

Riepilogo delle convenzioni

Riassumendo, le convenzioni sono le seguenti:

1. i pixel vengono trattati come aree e non come punti;
2. le coordinate intere della mesh partono col pixel (1,1) in alto a sinistra;
3. il valore numerico delle linee cresce verso il basso, quello delle colonne verso destra;
4. le coordinate intere e reali coincidono al centro del pixel;
5. le linee della griglia di coordinate vengono disegnate nei pixel che contengono il valore numerico della parte intera della linea, eccezion fatta per i bordi della mappa cartografica, le cui linee verranno disegnate sul pixel precedente.

3.6 Funzioni OAL utilizzate

Le funzioni appartenenti ad OAL, ed utilizzate per questa parte di progetto, rientrano in due sotto-gruppi differenti. Ve ne sono, infatti, per la gestione degli oggetti-cubo e altre che realizzano operazioni di generica utilità.

3.6.1 Gestione degli oggetti-cubo

OaParseLabelFile()

Utilizza il Livello Stream e L3 [cfr. 2.3.2] per leggere un file etichetta e acquisirla in memoria.

```
ODLTREE OaParseLabelFile(   char *filespec,
                            char *errfilespec,
                            MASK expand,
                            unsigned short nomsgs)
```

Il nome del file in ingresso è filespec, mentre errfilespec rappresenta il file su cui scrivere i messaggi d'errore in fase di parsing, nel caso che nomsgs abbia valore TRUE.
Restituisce un nodo dell'albero ODL.

OaReadImageFromQube()

Questa routine legge da un file cubo una singola banda e restituisce un oggetto di tipo immagine monocroma in memoria. Crea un albero ODL per gli attributi dell'oggetto e lo unifica ai dati dell'oggetto stesso.

```
OA_OBJECT OaReadImageFromQube(   ODLTREE qube_node,
                                 int band)
```

Per leggere le informazioni dal file è necessario passare il file cubo qube_node e il numero della banda desiderata band.
Restituisce un oggetto che può essere utilizzato dalle funzioni OAL per la gestione delle immagini.

3.6.2 Operazioni di generica utilità

OaReportError()

Scrive un messaggio d'errore su output standard che per default è il video. Tutte le routine OAL, verificatosi un errore, settano la variabile interna *oa_errno* per poi chiamare questa funzione.

```
int OaReportError(   char *error_message)
```

La stringa error_message è il messaggio che si vuole venga visualizzato, scritto su file o segnalato su altra periferica.
La funzione restituisce sempre 0.
Chiamando,invece, la routine `OaRouteErrorMessages()` si può indicare una nuova destinazione per i messaggi.

OaKwdValuetoLong()

Questa funzione trova uno specifico attributo nell'albero ODL e ne acquisisce il valore convertendolo in un dato di tipo 'long int'.

```
int OaKwdValuetoLong(   char *kwd_name,
                        ODLTREE odltreenode,
                        long *value)
```

L'attributo kwd_name viene cercato nel nodo dell'albero ODL odltreenode, convertito in

'long int' e memorizzato in value.
In caso di successo, la routine restituisce 0 e memorizza il valore dell'attributo. Se non riesce a fare la conversione, restituisce 1 e notifica l'errore. Se nel nodo ODL non è presente l'attributo, restituisce 1 senza alcun messaggio d'errore.

OaKwdValuetoStr()

Questa funzione trova l'attributo specificato in un nodo nell'albero ODL e setta il parametro value in modo da puntare direttamente ad esso.

```
int OaKwdValuetoStr(   char *kwd_name,
                       ODLTREE odltreenode,
                       char **value)
```

La stringa kwd_name specifica l'attributo da cercare nel nodo dell'albero ODL odltreenode. In caso di successo, la routine restituisce 0 e setta value in modo da puntare direttamente al punto dell'albero in cui è posto. Se nel nodo ODL non viene trovato l'attributo cercato, restituisce 1 senza alcun messaggio d'errore.

3.7 Procedura di importazione

In realtà, ci sono non una, bensì due procedure di importazione, le quali acquisiscono tipi differenti di dati. Esattamente, una serve per poter ottenere i file 'bmp' che andranno a comporre il database di partenza dell'adattatore, l'altra permette di creare dei file di testo con estensione 'pos' che contengono i dati di longitudine e latitudine dei file grafici creati. I due file differiscono l'un l'altro per la sola estensione: dall'immagine 'ui73n007.img' ricaveremo quindi i due file 'ui73n007.bmp' e 'ui73n007.pos'.
Per attivare la classe *OrthoPhoto_Package*, che si occupa proprio delle procedure di importazione, sulla riga di comando deve apparire la sequenza "`ultatools OP [option]`", dove *option* identifica la specifica procedura da attivare e al momento può assumere quattro valori:

- ? per avere una rapida guida in linea;
- N per importare le immagini NASA e creare il database di file 'bmp' per l'adattatore e, di conseguenza, per il PID;
- O per creare i file 'pos' con le indicazioni dei valori di latitudine e longitudine dei file immagine della NASA, necessarie per il completamento del database;
- P per adattare le immagini del database al sistema UltraPort.

Per il momento la nostra attenzione si focalizzerà sulla seconda e sulla terza delle opzioni or ora elencate. Per la quarta si invita a pazientare e leggere il capitolo 6 ad essa dedicato. Per avere un quadro esemplificato delle possibili strade che il programma può percorrere, conviene far riferimento alla figura 3.6, in cui la parte del *Selezionatore di immagini* viene interpretata proprio dalla classe OrthoPhoto_Package.

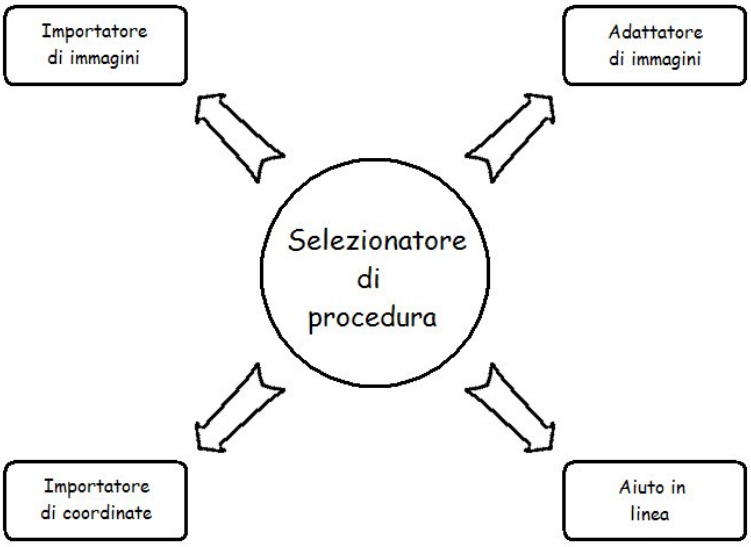

Figura 3.6. Funzionamento logico della classe OrthoPhoto_Package

Purtroppo, nonostante gli sforzi fatti, la sola via procedurale non ha potuto evitare la creazione di file di immagine che poi si sono dovuti correggere manualmente con programmi di fotoritocco. Difatti, alcune zone di un certo numero di file (circa il 10% del totale) presentavano delle colorazioni e riempimenti non perfettamente consoni alle zone buone con queste confinanti.

3.7.1 Importazione delle immagini

Comandi di avvio

L'importazione delle immagini si avvia inserendo nella riga di comando la dichiarazione

```
ultratools OP N <file> <par1> <par2> <par3>
```

in cui i parametri hanno il seguente significato:

<**file**> è il nome del file NASA (senza estensione) da importare.

<**par1**> indica la modalità con cui trattare i buchi nell'immagine e assume un ruolo importante per tutte quelle porzioni di immagini che risultano avere un'unica banda a disposizione, risultando sulle tonalità di grigio. Può assumere i valori:

- 0: nel caso si vogliano lasciare le immagini così come sono nel database originale della NASA;
- 1, 2, 3: per calcolare, rispettivamente, il fattore di scala medio, massimo o minimo con cui settare i valori RGB dei pixel delle aree in toni di grigio;

<**par2**> è un valore che serve per verificare che i valori RGB impostati per un determinato pixel grigio non si discostino troppo dai colori presenti sui bordi colorati del buco. Deve essere compreso tra 0 e 255.

<**par3**> è una stringa di due caratteri, che possono assumere i soli valori N o Y (No o Yes), e un intero compreso tra 0 e 21. Il primo permette di attivare le procedure per la manipolazione di file immagini con una tonalità molto scura di colori. Il secondo consente di attivare il riconoscimento e la modifica di macchie, presenti nei file NASA, dai colori del tutto fuori luogo come si evince dalla figura 3.7. Infine, l'intero indica il valore utile per regolare la soglia per il valore di riconoscimento dei pixel buco: più è alto, più bassa risulta la soglia.
Per maggiori dettagli si faccia riferimento alle sezioni 4.5, 4.7, 4.8 e 4.9.

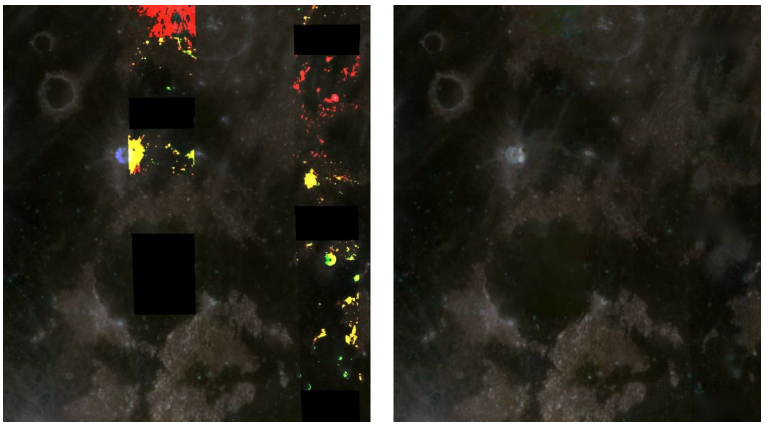

Figura 3.7. Un'immagine scura con buchi e chiazze colorate a confronto con la medesima una volta ultimata l'importazione

Per importare, ad esempio, il file 'ui03s333.img', la sequenza di comando da immettere è la seguente:

```
ultratools OP N ui03s333 2 0 YY3
```

Per quanto esposto, si può evincere che, per come è stata creata l'interfaccia, si può operare su un solo file alla volta.
Per evitare di scrivere ripetutamente la riga di comando, è comodo avere un file batch con al suo interno tutte le chiamate necessarie per far eseguire il programma su tutti i 996 file del database della NASA. A tale scopo si è creata una classe che permettesse di scrivere in modo automatico un file batch che vedremo più avanti [cfr. sezione 3.8].

Funzionamento

La sequenza delle operazioni necessarie all'importazione, sebbene possa apparire lunga e complessa, in realtà risulta abbastanza lineare anche se alle volte un po' tortuosa.
Il punto di partenza dell'intera procedura è `OrthoPhoto_Package::bProcessCommandLine()`,

funzione che, dopo aver controllato la correttezza dei parametri in ingresso, avvia l'acquisizione dei dati con una chiamata alla procedura `Filesystem_Package::UserProcessFile()`. Questa ha sostanzialmente tre compiti:

1. cerca nel database della NASA il file 'jpg' a massima definizione corrispondente al nome del file dato in input, acquisendo l'immagine stessa;
2. acquisisce i dati dal file 'img' della NASA chiamando `OrthoPhoto_Nasa::GetData()`;
3. setta e crea il file 'bmp' da utilizzare per generare il database da utilizzare in input per l'adattatore.

Evidentemente, è `OrthoPhoto_Nasa::GetData()` ad assurgere un ruolo primario. Essa, infatti, si occupa di tutta la parte che riguarda la manipolazione dei dati.
In primis, combina, grazie a `OrthoPhoto_Nasa::MergeData()`, i dati derivanti dal 'jpg', per mezzo di `OrthoPhoto::OrthoPhotoFromJpg()`, con quelli 'img', che hanno una risoluzione doppia rispetto ai primi. Lo scopo è quello di ottenere immagini con colorazione analoga a quella NASA e con il massimo livello di dettaglio disponibile.
Una volta ottenuta l'immagine in memoria, questa viene manipolata con le funzioni per la ricerca, l'identificazione e la correzione di eventuali buchi neri, rispettivamente `OrthoPhoto_Nasa::FindHole()`, `OrthoPhoto::HoleChecking()` e `OrthoPhoto::HoleSetting()`. I buchi neri sono zone della superficie lunare che la NASA non è stata in grado di disegnare nei suoi file grafici [cfr. sezione 4.2]. Le funzioni citate funzionanao anche per le aree grigie e le placche colorate [cfr. sezione 4.8].
Infine, con l'immagine 'bmp' ormai pronta e corretta, non rimane altro che scrivere il file con `OrthoPhoto::OrthoPhotoTestOut()`.

3.7.2 Importazione delle coordinate

Comandi di avvio

Per l'importazione delle coordinate geografiche delle immagini vale un discorso analogo a quanto descritto per le immagini e si avvia inserendo nella riga di comando la dichiarazione

```
ultratools OP 0 <dir>
```

in cui l'unico parametro ha il seguente significato:

<**dir**> è la directory dove risiedono tutti i file immagine del database NASA che compongono il mosaico e di cui bisogna creare i file 'pos'.

Funzionamento

Proprio come per le immagini, è `OrthoPhoto_Package::bProcessCommandLine()` che si preoccupa di avviare l'intera procedura di importazione. Stavolta, però, viene richiamata la procedura `Filesystem_Package::ProcessFiles()`.
Questa permette di ricercare, uno ad uno, tutti i file etichetta 'lab' presenti nella directory specificata in input e di creare, tramite `OrthoPhoto_Nasa::GetCoord()`, un file 'pos' che contiene le dimensioni in pixel del file immagine e la sua estensione, definita

attraverso i quattro punti agli angoli dell'immagine espressi come coppia di coordinate (*longitudine*,*latitudine*) in radianti.

3.8 Creazione del file batch di base

3.8.1 Generalità

La procedura di importazione dei file grafici agisce su un file alla volta. Per rendere più veloce l'intero procedimento, senza dover inserire una ad una le righe di comando per i 996 file, si è pensato di creare un file batch contenente le 996 chiamate singole per attivare l'importazione.
Si sarebbe potuto procedere manualmente, ma per comodità e per eventuali utilizzi futuri, si è creata la classe *Utility_Package* che in maniera automatica crea questa lunga serie di chiamate, tutte con gli stessi parametri.
Una volta creato il file, basterà copiarlo nella directory dove è presente l'eseguibile di 'ultratools.exe' e l'importazione avrà inizio.

Comandi di avvio

Per avviare la procedura di creazione del file batch, nella riga di comando è necessaria una dichiarazione del tipo

```
ultratools UT B CLEM <file> <dir> <par1> <par2> <par3>
```

in cui i parametri hanno il seguente significato:

<**file**> è il nome del file batch (senza estensione) da creare.

<**dir**> è la directory dove cercare i file di cui bisognerà creare la riga di comando.

<**par1**> indica la modalità con cui trattare i buchi nell'immagine e assume un ruolo importante per tutte quelle porzioni di immagini che risultano avere un'unica banda a disposizione, risultando sulle tonalità di grigio. Può assumere i valori:

- 0: nel caso si vogliano lasciare le immagini così come sono nel database originale della NASA;
- 1, 2, 3: per calcolare, rispettivamente, il fattore di scala medio, massimo o minimo con cui settare i valori RGB dei pixel delle aree in toni di grigio;

<**par2**> è un valore che serve per verificare che i valori RGB impostati per un determinato pixel grigio non si discostino troppo dai colori presenti sui bordi colorati del buco. Deve essere compreso tra 0 e 255.

<**par3**> è una stringa di due caratteri, che possono assumere i soli valori N o Y (No o Yes), e un intero compreso tra 0 e 21. Il primo permette di attivare le procedure per la manipolazione di file immagini con una tonalità molto scura di colori. Il secondo consente di attivare il riconoscimento e la modifica di macchie, presenti nei file NASA, dai colori del tutto fuori luogo come si evince dalla figura 3.7. Infine,

l'intero indica il valore utile per regolare la soglia per il valore di riconoscimento dei pixel buco: più è alto, più bassa risulta la soglia.
Per maggiori dettagli si faccia riferimento alle sezioni 4.5, 4.7, 4.8 e 4.9.

In fin dei conti la riga di comando è molto simile a quella utilizzata proprio per l'importazione dei file immagine.

3.8.2 Spiegazione dell'algoritmo

La prima cosa effettuata è un controllo: secondo le specifiche prima descritte, una riga di comando senza gli ultimi tre parametri viene accettata, ma senza l'indicazione del nome del file e della directory, è prevista l'uscita dal programma con la visualizzazione di una concisa guida in linea, chiamata dalla procedura `Utility_Package::SpecUtilityInfo()`, con una breve esposizione dei comandi corretti da utilizzare.
Se la riga di comando è corretta, tutti quei parametri in essa non specificati verranno settati con dei valori di default: 0 per par1, 40 per par2 e NN4 per par3. Altrimenti, i valori specificati verranno memorizzati, rispettivamente, nelle variabili membro s16AcquisitionMode, s16ColorDelta e pszImgOption.
Ora viene chiamata la procedura `Utility_Package::ProcessFiles()` che si occupa di cercare i file all'interno della directory specificata e passata come parametro. Per ogni file trovato, viene richiamata `Utility_Package::UserProcessFile()`. È proprio quest'ultima che effettivamente va a scrivere la riga di comando nel file batch, non prima di aver verificato l'esatta corrispondenza tra i file cercati e quelli trovati. Così, per ogni file immagine trovato nella cartella di ricerca, viene creata un'apposita riga di comando che viene scritta nel file batch.
Per la descrizione particolareggiata delle routine qui utilizzate si rinvia, in particolare, alla sezione 5.6.

Capitolo 4

Algoritmi dell'importatore

4.1 Vista d'insieme

Per poter importare i dati in modo da raggiungere l'obiettivo finale, di volta in volta si è dovuto capire il problema a cui si andava incontro e scegliere la strada migliore per risolverlo.
I problemi affrontati sono stati parecchi, in quanto, per stessa ammissione dei tecnici della NASA, i dati a disposizione sono di difficile utilizzo e, alle volte, risultano non completi o poco affidabili. Questi dati, infatti, prima di poter essere utilizzati per la creazione del database del PID hanno dovuto essere manipolati, alle volte anche pesantemente, sia proceduralmente sia manualmente.
In questo capitolo si parlerà, in modo diffuso e logico, delle operazioni utili allo scopo per poi addentrarsi, in modo più particolareggiato, negli algoritmi che permettono di realizzarle a livello pratico e quindi procedurale. Gli argomenti trattati sono molteplici e riguardano i due importanti filoni di acquisizione dei dati e della loro manipolazione.

4.2 Acquisizione dei dati dai file 'img'

Nei file 'img' sono presenti i dati puri della missione. Si trovano, perciò, i valori di cinque bande elettromagnetiche afferenti alla camera UVVIS [cfr. paragrafo 2.2.2] a diverse frequenze portanti: 415 nm, 750 nm, 900 nm, 950 nm, 1000 nm.
Secondo la documentazione della NASA e in accordo con lo spazio dei colori RGB, si dovrebbero utilizzare la banda a 415 nm per il colore blu, 750 nm per il verde e 950 nm per il rosso, così da avere un ritratto della Luna in toni quasi naturali. Purtroppo, dopo aver fatto svariati tentativi di acquisizione di immagine con tali dettami, sono risultati due problemi:

- i colori risultavano abbastanza lontani da quelli visibili nell'album lunare, incluso nel pacchetto software, basato sulle immagini 'jpg', create probabilmente ad arte dagli ingegneri NASA;

- quasi il 40% delle immagini risultanti, ed esattamente 395 su 996, conteneva al suo interno dei buchi.

Per risolvere tali problemi, si è proceduto acquisendo sia le immagini 'jpg' che i dati dei file 'img', per poi metterli insieme.

4.2.1 Definizione di buco, macro-buco e pixel-buco

Per chiarire il concetto di buco in tale ambito, partiamo con un esempio: si prenda in considerazione un puzzle e si inizi a comporlo fino a quando non finiscono i pezzi. A questo punto si può verificare se il puzzle è completo o meno. Se per un qualsiasi motivo dovesse mancare anche solo un pezzo, si avrebbe davanti agli occhi un puzzle con un buco. Più pezzi mancanti adiacenti, invece, formerebbero un grande buco.
Un'immagine è paragonabile ad un puzzle in quanto è composta di tanti pixel, ovvero i pezzi. Nel nostro caso un buco è una lacuna, colmata dal colore nero, di una porzione di immagine. Questa area nera è composta di pixel-buco che, essendo adiacenti, formano un macro-buco (o più semplicemente buco).
Sostanzialmente, un pixel che ha componenti (R,G,B) = (0,0,0) è da considerarsi pixel-buco. Un macro-buco è composto da almeno due pixel-buco adiacenti. La figura 4.1 può risultare d'aiuto.

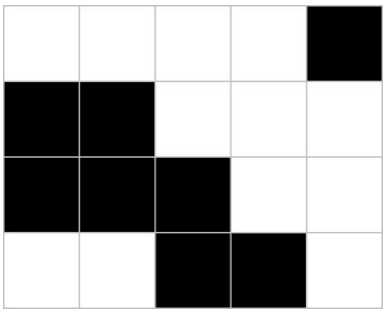

Figura 4.1. Macro-buco e pixel-buco in un ipotetica griglia di pixel

Spiegazione dell'algoritmo

L'acquisizione dei dati dai file 'img' avviene all'interno di `OrthoPhoto_Nasa::GetData()`, prima dell'inizio di tutte le altre operazioni.
Per poter acquisire i dati, in primo luogo è stato necessario aprire il file etichetta e leggere dallo stesso una serie di valori dell'oggetto-cubo come le dimensioni dei suoi assi, in quanto queste permettono di capire la larghezza e l'altezza dell'immagine rappresentata nonché il numero dei bande spettrometriche a disposizone.
Una volta fatto ciò, e dopo aver aperto in lettura il file 'img' con la chiamata alla funzione `OaReadImageFromQube()` della libreria OAL, si è fatta una scansione, una banda alla volta,

dei dati, immagazzinandoli in vettori. In sostanza, alla fine della scansione, ci si è trovati con cinque vettori ognuno dei quali contenente i dati di una e una sola banda spettrale.

4.3 Acquisizione dell'immagine 'jpg'

Dopo aver acquisito i dati dal cubo, è necessario leggere e memorizzare l'immagine 'jpg' ad essa corrispondente. Per farlo si chiama la procedura `OrthoPhoto::OrthoPhotoFromJpg()`.

Spiegazione dell'algoritmo

Il funzionamento è abbastanza semplice e lineare, in quanto la procedura si occupa di aprire il file 'jpg' in questione, di leggere i valori dei pixel e di trasferirli in tre vettori, corrispondenti alle tre componenti cromatiche RGB.

4.4 Fusione tra 'img' e 'jpg'

La procedura `OrthoPhoto_Nasa::MergeData()` realizza quello che è il passo successivo: unire i dati del file 'img' con quelli del 'jpg'.
L'idea di base è quella di creare delle immagini con la massima risoluzione possibile, corrispondente a quella dei file 'img'. Purtroppo rispetto a questi, i file 'jpg' risultano avere dimesioni dimezzate sia in larghezza sia in altezza. Conseguentemente, ad ogni pixel 'jpg' corrispondono quattro pixel 'img', come mostrato in figura 4.2 dove in nero sono delimitati i pixel 'jpg' ed in grigio quelli 'img'. Senza una qualche trasformazione, si otterrebbero, perciò, immagini dalle dimesioni massime ma con blocchetti quadrati di 2 pixel per lato: una specie di macro-pixel che potrebbe risultare sgradevole alla vista.

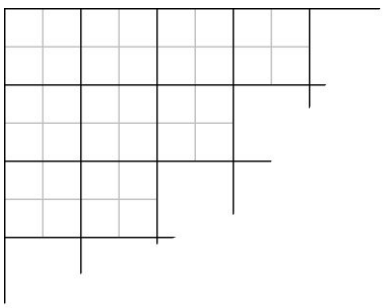

Figura 4.2. Corrispondenza tra pixel 'jpg' e quelli 'img'

Si è pensato, dunque, di prendere questo macro-pixel e alterare i valori dei 4 pixel che lo compongono in modo adeguato. Utilizzando i valori dei dati 'img' per calcolare una sorta di media quadratica, si è potuto intervenire direttamente sui valori RGB dei pixel.

Tale procedura di integrazione di due tipi di file diversi, ha permesso di recuperare intere aree che nell'album lunare della NASA comparivano nere.
Per la creazione dei file 'jpg' gli ingegneri americani si sono basati sull'utilizzo di sole tre delle cinque bande elettromagnetiche a loro disposizione. Di molte di tali zone si sono potuti recuperare i dati dalle rimanenti due bande.
Non avendo però un corrispettivo file a colori da cui acquisire le tonalità cromatiche, ci si è dovuti accontentare, come passaggio intermedio, di impostare i canali RGB di tali zone con lo stesso valore numerico, rendendole visivamente sulla tonalità di grigio, e solo in un secondo momento si è provveduto a dar loro una colorazione congrua.

4.4.1 Definizione cromatica di un pixel

Abbiamo capito [cfr. sezione 3.2] che le immagini utilizzate aderiscono al modello RGB. Ora è doveroso specificare che ogni pixel, per poter aderire a tale standard, deve necessariamente avere tre valori, rappresentanti le componenti cromatiche, che variano nell'intervallo numerico da 0 a 255, dove 0 rappresenta l'assenza di luce, ovvero il colore nero, e 255 la massima presenza della stessa, e quindi il colore bianco.
Combinando i valori dei tre canali, l'occhio umano percepisce un colore piuttosto che un altro. Ogni combinazione (R,G,B), pertanto, corrisponde ad un preciso colore: (0,0,0) al nero, (255,255,255) al bianco, (255,0,0) al rosso, (0,255,0) al verde, (0,0,255) al blu, (255,255,0) al giallo, (255,0,255) al magenta, (0,255,255) al ciano. Naturalmente, combinando opprtunamente i valori, si possono ottenere tutti i colori dello spettro visivo.

Spiegazione dell'algoritmo

La funzione di accorpamento prevede che si faccia la scansione dei file 'img' pixel per pixel. Ognuno di essi è identificato da una coppia di numeri (i,j) indicanti, rispettivamente, l'indice di colonna e quello di riga. Solitamente la scansione avviene per righe: per una data riga j si fa variare l'indice i fino alla fine della medesima, per poi passare a quella successiva. In base a tale coppia di valori, si calcolano altri due indici che permettono di capire a quale pixel 'jpg' ci si riferisce. Il calcolo è dato da una semplice divisione per due e (m,n) è la nuova coppia di indici referenti al 'jpg'.

$$m = \left\lfloor \frac{i}{2} \right\rfloor \quad ; \quad n = \left\lfloor \frac{j}{2} \right\rfloor$$

Per capire meglio la corrispondeza tra i due formati di immagine, e quindi tra i diversi valori degli indici di riga e colonna, si può fare riferimento alla figura 4.3, dove in nero sono rappresentati i pixel 'jpg' e in grigio quelli 'img'.
Ora che si è individuata la corrispondenza tra pixel, è necessario fondere i valori RGB di uno con quelli dell'altro. A tale scopo, si è pensato di utilizzare i dati numerici dei pixel 'img' per modificare leggermente la tinta di base proveniente dai corrispettivi pixel 'jpg'. Questa procedura serve per evitare che l'immagine risultante sia composta da blocchi di pixel 2×2 con lo stesso colore. Infatti, per come pensato e realizzato, tale algoritmo modifica i valori RGB di un pixel, quel tanto che basta ad evitare il formarsi di macro-pixel.

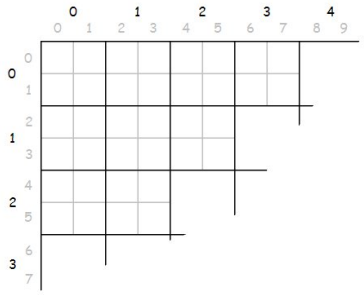

Figura 4.3. Corrispondenza tra indici 'jpg' e quelli 'img'

L'algoritmo segue, sostanzialmente, tre passi.

1. Nel primo viene calcolata la media dei valori di banda presenti per quel dato pixel 'img' secondo la formula

$$Intensity = \frac{\sum_{i=1}^{5} Bands_i}{Bands_{number}}$$

   dove $Bands_i$ è il valore numerico della i-esima banda spettrometrica e $Bands_{number}$ è il numero di bande che apportano un valore diverso da zero alla sommatoria.

2. In secondo luogo, si calcola l'intensità totale dei quattro pixel 'img' che, idealmente, sono coperti dal pixel 'jpg'

$$Intensity_{tot} = \frac{\sum_{i=1}^{4} Intensity_i}{4}$$

   Risulta chiaro che $Intensity_i$ si riferisce all'intensità del pixel i-esimo, così come calcolata al passo precedente.

3. Infine, viene calcolato una sorta di fattore di normalizzazione

$$K = \frac{Intensity}{Intensity_{tot}}$$

   per poi applicarlo alle componenti RGB del singolo pixel 'img' come segue

$$R_{img} = K \cdot R_{jpg} \quad ; \quad G_{img} = K \cdot G_{jpg} \quad ; \quad B_{img} = K \cdot B_{jpg}$$

Una nota importante: nell'accezione generale, quando si manipolano delle immagini, ci si riferisce ai loro pixel tramite una coppia di valori formata da indici di riga e colonna,

ma è solo una comodità cognitiva. Per come è strutturato l'intero programma e tutte le librerie ad esso afferenti, nel cuore del codice si utilizzano gli indici in maniera linearizzata. Per capire, un'immagine di grandezza 4×3 viene memorizzata in un vettore lineare di 12 elementi e non in una matrice 4×3, come mostrato in figura 4.4. Quindi, dietro ogni individuazione di pixel tramite una coppia di indici, c'è in realtà anche il calcolo del corrispettivo indice lineare per la corretta manipolazione del pixel.

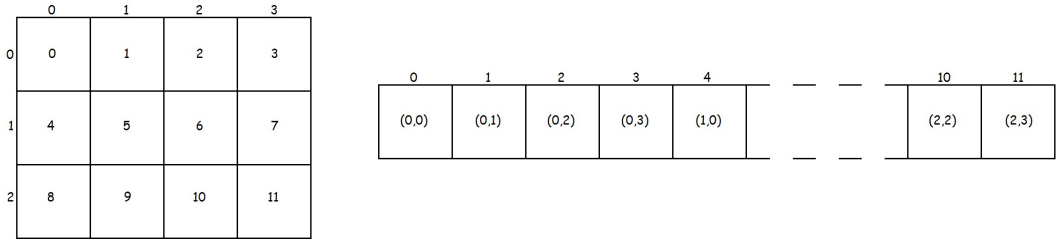

Figura 4.4. Esempio di relazione tra indici matriciali e quelli linearizzati

4.5 Ricerca ed identificazione dei buchi

Una volta acquisita, in memoria residente si ha un'immagine completa alla risoluzione voluta. Molto spesso, come detto precedentemente, ci si imbatte in file grafici con imperfezioni come buchi neri o zone grigie. Con le procedure `OrthoPhoto_Nasa::FindHole()` e `OrthoPhoto::HoleChecking()` è possibile individuare non solo i pixel che rappresentano un buco ma anche quelli che fanno parte delle zone grigie, e poi assegnare loro un indice numerico, identificativo del macro-buco o della zona grigia di cui fanno parte.
Dopo vari tentativi, si è capito che era necessario impostare un valore di soglia sui canali RGB che permettesse di distinguere tra pixel buoni e quelli buco, in quanto nei file grafici della NASA i pixel-buco avevano anche valori diversi da zero.
I pixel inerenti alle aree grigie, invece, vengono segnalati con una variabile booleana in fase di acquisizione dell'immagine.
Le procedure qui indicate, vengono utilizzate anche in fase di riconoscimento delle placche colorate [cfr. sezione 4.8].

4.5.1 Impostazione del valore di soglia

Osservando un gran numero di file grafici e dopo averne provato di diversi, si è deciso che il miglior valore numerico di soglia fosse uguale a 20. Tale limite ha permesso, per la grande maggioranza di file, di riconoscere correttamente i pixel-buco.
Purtroppo si è anche constatato che per alcune immagini dai toni complessivamente troppo scuri, tale limite risultava fin troppo elevato tanto da considerare buchi anche pixel che in realtà non lo erano. Per risolvere questo inconveniente, si è utilizzato il numero presente nel terzo parametro nella stringa di comando [cfr. paragrafo 3.7.1], in modo che il valore

di soglia impostato di default venisse diviso per quello del parametro, così da abbassare la soglia stessa.

Spiegazione dell'algoritmo

In `OrthoPhoto_Nasa::FindHole()`, nella modalità di ricerca di buchi, si fa un semplice ciclo su tutti i pixel dell'immagine. La procedura `OrthoPhoto::HoleChecking()` viene chiamata se il pixel in analisi non è stato già trasformato in uno buono e solo se è uno di quelli che forma una zona grigia o se almeno due su tre delle sue componenti RGB risultano inferiori al valore di soglia.
Non appena viene trovato un pixel con le caratteristiche appena descritte, inizia la procedura che assegna un codice identificativo univoco ad ognuno dei macro-buchi o delle zone grigie.
Il ruolo di `OrthoPhoto::HoleChecking()` è quello di far partire una scansione che permette di dare lo stesso codice identificativo a pixel appartenenti alla stessa zona dello stesso tipo. Questo avviene tramite una semplice variabile contatore e una lista di interi in cui è possibile memorizzare l'indice lineare dei pixel interessati.
La scansione segue una logica precisa: a partire dal pixel individuato, si controllano anche gli otto che lo circondano che, se congruenti alle specifiche, verranno marchiati come buchi, facendo incrementare il contatore. Il valore del codice identificativo per i pixel buoni è 0, mentre per i buchi si parte dal numero 1 in avanti, come mostrato in figura 4.5 in cui sono stati omessi i codici dei pixel buoni.

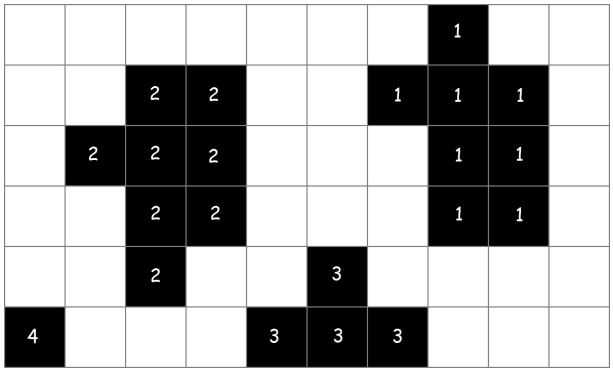

Figura 4.5. Ipotetica identificazione di macro-buchi

L'algoritmo prevede anche un limite massimo di gestione di 80 macro-buchi, in quanto si è verificato che senza di esso, alcune immagini particolarmente "bucate" superassero di gran lunga tale limite portando ad una esagerata occupazione della memoria ed, alle volte, anche alla prematura e fallace fine dell'esecuzione del programma.

4.6 Riconoscimento dei bordi dei macro-buchi

Prima di intervenire sui buchi, è essenziale fare un'operazione: cercare ed identificare i bordi di ogni macro-buco o delle zone grigie. Tale compito viene assunto dalla procedura `OrthoPhoto::HoleSetting()`, impostando un opportuno valore parametrico nella chiamata alla stessa. Per bordo si intende una specie di cornice di pixel buoni che cingono un macro-buco (o una zona grigia).
Tale bordo risulta fondamentale per il riempimento dei buchi [cfr. sezione 4.7] e per la correzione delle aree grigie nonché delle placche [cfr. sezione 4.9], in quanto le informazioni cromatiche, per le operazioni di correzione dei vari tipi di pixel, vengono dedotte proprio da esso.
I dati riguardanti il bordo, ovvero i pixel con le loro componenti RGB, vengono salvate in una struttura in memoria che viene gestita dinamicamente all'occorrenza, in modo da non appesantire le altre fasi del programma.

Spiegazione dell'algoritmo

Prima di procedere a tutte le operazioni correttive, ma dopo aver individuato i buchi presenti e averne contato il numero, in `OrthoPhoto_Nasa::GetData()` si procede ad una scansione dell'immagine pixel per pixel. Se uno di essi viene trovato con codice identificativo compreso tra quelli in corso di manipolazione, ovvero nell'intervallo tra 1 e 79, viene chiamata `OrthoPhoto::HoleSetting()` nella modalità che permette di individuare i bordi dei macro-buchi.
All'interno di tale procedura e a partire dal pixel individuato, si controllano quelli ad esso contigui. Se almeno uno di quest'ultimi è identificato dal codice 0, siamo in presenza di un pixel buono. Naturalmente, si può capire che un pixel così identificato e affiancato ad uno di tipo buco, rappresenta un pixel di bordo.
Si va, allora, alla ricerca di tutti i pixel buoni attorno a quest'ultimo, con una escursione massima di 5 pixel nelle quattro direzioni possibili. In sostanza, viene considerato un quadrato di 11 pixel per lato, con al centro il pixel di bordo individuato.
Ora si scansiona questo quadrato e tutti i pixel buoni vengono presi in considerazione memorizzando il loro indice lineare nella lista di pixel afferenti al bordo per quel dato macro-buco.
Nella figura 4.6 si è cercato di spiegare tale metodo. Il pixel identificato dalla lettera A è il pixel buono individuato che risulta al centro del quadrato di pixel di bordo segnati con la lettera B. Come si vede, i pixel-buco, qui semplicemente colorati in nero, non vengono presi in considerazione come bordo. Inoltre si evince che il quadrato non è di 11 pixel per lato, in quanto al di sopra del pixel A ci si può estendere per soli 3 pixel, dato che poi si perviene alla fine dell'immagine.
Alla fine della scansione totale dei pixel dell'immagine, si avranno tante liste di pixel di bordo quanti sono i macro-buchi.

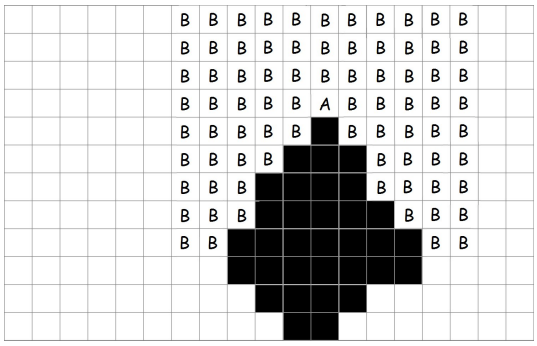

Figura 4.6. Identificazione dei bordi nei macro-buchi

4.7 Riempimento dei buchi

Le operazioni di riempimento dei buchi e di colorazione delle aree grigie sono affidate alla procedura `OrthoPhoto::HoleSetting()`, chiamata con una modalità differente rispetto all'utilizzo che se ne fa per l'individuazione dei bordi.
Prima di poter effettivamente correggere le immagini, è però opportuno sapere che precedentemente alla sua chiamata, che comunque avviene all'interno di un ciclo su tutti i pixel dell'immagine, vengono fatte alcune operazioni utili al suo funzionamento.
Senza entrare troppo nel merito, in quanto leggendo direttamente il codice se ne può apprendere abbastanza facilmente la logica, ci si preoccupa, in primo luogo, di calcolare un colore medio per ogni macro-buco individuato. Secondariamente, ma non per importanza, si stabiliscono i valori massimi e minimi RGB per quel che concerne le strutture che rappresentano i pixel di bordo.
Tali informazioni risulteranno assai utili e a breve ne capiremo i motivi.

4.7.1 La struttura dati per i pixel di bordo

Per una maggiore comprensione dell'algoritmo, è bene focalizzare brevemente la nostra attenzione sulla struttura dati che consente di memorizzare le informazioni utili per la correzione dei buchi.

```
struct m_SMacroHoleStruct{   bool *pbEdgePix, bGreyPix;
                             unsigned long u32TotEdgePix, *pu32EdgePixList;
                             unsigned long v3u32SumColor[3], u32InnerIndex;
                             unsigned int v3u16CompMax[3], v3u16CompMin[3];
                             double v3f64AverageColor[3]; };
```

Nella struttura sono presenti semplici variabili ma anche diversi tipi di vettori, che in fin dei conti assurgono un ruolo fondamentale per differenti aspetti.
Il vettore dinamico booleano pbEdgePix è grande tanto quanti sono i pixel dell'immagine e segnala se il corrispettivo pixel è da considerare come bordo di un macro-buco; in tal caso in pu32EdgePixList ne viene memorizzato l'indice lineare.

Per il bordo di ogni macro-buco è assai rilevante memorizzare la somma dei tre canali cromatici nel vettore di interi v3u32SumColor, nonché i valori di minimo e massimo dei singoli dati RGB, trovati durante la scansione dei pixel di bordo, nei due vettori v3u16CompMin e v3u16CompMax.
Infine, il vettore v3f64AverageColor, come vedremo, servirà per memorizzare le componenti RGB del colore medio derivato dalle informazioni di bordo a nostra disposizione.

Spiegazione dell'algoritmo

Prima di modificare adeguatamente i pixel, occorre impostare i parametri della struttura m_SMacroHoleStruct calcolandone i valori chiariti in precedenza.
L'operazione di somma dei valori RGB viene fatta semplicemente scorrendo la lista dei pixel di bordo e sommando i valori di volta in volta. Terminata la scansione dei pixel di bordo dei macro-buchi a disposizione, è facile ricavare un valore medio per ogni singola componente cromatica secondo il banale calcolo

$$v3f64AverageColor[i] = \frac{v3u32SumColor[i]}{u32TotEdgePix} \quad , \quad i = 1,2,3.$$

A questo punto, sempre in dipendenza dei macro-buchi a disposizione, si esegue una scansione delle strutture dei bordi e si calcolano i valori di minimo e massimo dei singoli canali RGB secondo dei semplici raffronti numerici.
Ora si inizia una scansione di tutti i pixel dell'immagine e se il proprio codice identificativo risulta tra quelli che identificano un buco (ovvero tra 1 e 79), allora viene chiamata la funzione `OrthoPhoto::HoleSetting()`, con la modalità per la correzione dei buchi.
Il mezzo con cui è possibile riempire i macro-buchi, o correggere le zone grigie con una colorazione adeguata, è quello di calcolare una sorta di media pesata in relazione alla posizione del pixel, in termini di distanza dal bordo, all'interno del macro-buco o della zona grigia, dualismo che ora si scinderà, in quanto le procedure di correzione sono differenti tra loro.
Genericamente parlando la media pesata è definita come

$$\overline{v} = \frac{\sum_{i=1}^{n} p_i \cdot v_i}{\sum_{i=1}^{n} p_i}$$

Nel nostro caso, v_i è il valore della componente cromatica dell'i-esimo pixel (x_i,y_i) preso in considerazione e il peso p_i risulta essere il quadrato dell'inverso del quadrato della distanza da ogni pixel di bordo (x_b,y_b) secondo quanto segue

$$p_i = \left(\frac{1}{D^2}\right)^2 \qquad , \qquad D^2 = (x_i - x_b)^2 + (y_i - y_b)^2$$

Infine, per ricondurci alla terminologia utilizzata nella stesura del codice, il numeratore

della media pesata è stato denominato $f64ColorQuadDistanceSum[j]$, mentre il denominatore $f64TotInverseQuadDistance$. Risulta quindi che la nostra media pesata MP_v sarà

$$MP_v \equiv \overline{v} = \frac{f64ColorQuadDistanceSum[j]}{f64TotInverseQuadDistance} \quad , \quad j = 1,2,3$$

Il numeratore possiede un indice in quanto rappresenta le tre componenti RGB relative al pixel da correggere.
Si procede, quindi, al calcolo dei valori delle componenti RGB, applicando il criterio della media pesata utilizzando per tre volte la stessa formula, differenziandola solo rispetto alle componenti stesse:

$$R_p = \frac{f64ColorQuadDistanceSum[1]}{f64TotInverseQuadDistance} \equiv \frac{\sum_{i=1}^{n} p_i \cdot R_i}{\sum_{i=1}^{n} p_i}$$

$$G_p = \frac{f64ColorQuadDistanceSum[2]}{f64TotInverseQuadDistance} \equiv \frac{\sum_{i=1}^{n} p_i \cdot G_i}{\sum_{i=1}^{n} p_i}$$

$$B_p = \frac{f64ColorQuadDistanceSum[3]}{f64TotInverseQuadDistance} \equiv \frac{\sum_{i=1}^{n} p_i \cdot B_i}{\sum_{i=1}^{n} p_i}$$

Riempimento dei macro-buchi

Per i pixel considerti buchi, e quindi che appaiono neri alla vista, l'algoritmo non prevede ulteriori calcoli, in quanto si hanno a disposizone tutti gli elementi utili allo scopo e i valori R_p,G_p,B_p precedentemente calcolati sono proprio le componenti RGB che andranno a sostituire i corrispettivi valori del pixel-buco.
Quindi

$$R = R_p \quad ; \quad G = G_p \quad ; \quad B = B_p$$

Colorazione della aree grigie

Per le aree grigie il discorso è lievemente più complicato, poiché non si possono banalmente sovrascrivere i valori RGB del pixel, ma bisogna tenerne conto per non perdere la relazione fra tutti i pixel adiacenti.
Infatti le peculiarità del terreno lunare sono date dalle differenti tonalità di grigio di pixel contigui, ed è proprio tale relazione da dover essere tenuta in considerazione e sfruttata a dovere. In sostanza, esistono già dei valori RGB buoni (che permettono di non far considerare il pixel come buco) tutti e tre numericamente uguali.
Essenzialmente, per il calcolo delle componenti RGB dell'i-esimo pixel si usano le seguenti formule:

$$R = K_R \cdot R_i \quad ; \quad G = K_G \cdot G_i \quad ; \quad B = K_B \cdot B_i$$

I fattori K_R,K_G,K_B servono per scalare i volori cromatici partendo da quelli originali e

senza perdere le relazioni con i medesimi valori dei pixel contigui e, prendendo ad esempio una sola delle tre componenti, sono calcolati come segue:

$$K_R = \frac{R_p}{f64PivotColor}$$

$f64PivotColor$ è una variabile che può essere calcolata in tre modi differenti, scelta che viene impostata nella riga di comando [cfr. paragrafo 3.7.1]. Quindi, il programma viene predisposto all'utilizzo di una delle tre formule in base al parametro par1 che assume il valore 0, in caso non si vogliano modificare le immagini originali, oppure variare da 1 a 3, volendole correggere.

- Con par1=1 viene calcolato un fattore medio prendendo in considerazione le tre componenti dell'i-esimo pixel.

$$f64PivotColor = \frac{1}{3} \cdot \sum_{i=1}^{3} \frac{f64ColorQuadDistanceSum[i]}{f64TotInverseQuadDistance} \equiv \frac{R_p + G_p + B_p}{3}$$

- par1=2 come valore parametrico indica, invece, di calcolare il massimo confrontando i valori della tripletta (R_p,G_p,B_p).
- par1=3, infine, se è necessario cercarne il minimo.

In conclusione, sempre prendendo in esame una sola delle tre componenti e partendo dall'i-esimo pixel, risulterà

$$R = \frac{R_p \cdot R_i}{f64PivotColor}$$

4.8 Ricerca delle placche

In alcuni file immagine della NASA, dalle tonalità molto scure, capita di trovare delle chiazze colorate con colori troppo netti che nulla hanno da condividere con le tonalità cromatiche dell'immagine stessa: appaiono, così, chiazze (in seguito chiamate anche placche) rosse, verdi, gialle e blu. Per capire, si faccia riferimento alla figura 3.7.
Probabilmente, quando la NASA ha creato i suoi file 'jpg', è andata incontro a piccoli errori nel calcolo dei valori delle componenti RGB dei pixel che rientrano in tale placca.
Sfruttando le capacità degli algoritmi fin qui presentati, con l'aggiunta di qualche particolarità, si è potuto correggere anche tale tipo di errore grafico. In fin dei conti, invece di andare a trovare una zona formata di tanti pixel-buco, si cercano pixel con colori troppo difformi dalla tonalità generale dell'immagine che si scandisce.
Tali placche si presentano solo in alcuni file scuri: la prima distinzione che si è dovuta fare, perciò, è quella tra file semplicemente scuri e file scuri con placche, con buchi o senza non ha importanza.
Anche in questo caso risulta discriminante la riga di comando ed in particolare il parametro par3 che racchiude in sé due caratteri che consentono di capire se si è in presenza di

un file scuro e con le chiazze [cfr. paragrafo 3.7.1].
Dopo aver corretto le immagini con tonalità cromatiche non particolarmente scure, si riparte, in buona sostanza, con le stesse procedure viste per i macro-buchi, andando a manipolare, però, solo le immagini scure ed eventualmente con la placche.

Spiegazione dell'algoritmo

Per cercare le placche si usano le stesse routine utilizzate per i macro-buchi e che operano in modo analogo: `OrthoPhoto_Nasa::FindHole()` e `OrthoPhoto::HoleChecking()`. La differenza sta nell'attivare degli algoritmi diversi grazie all'identificazione, tramite la riga di comando, dell'immagine scura e "placcata".
Ecco, dunque, che in `OrthoPhoto_Nasa::FindHole()` si andranno a cercare i pixel che hanno almeno uno dei tre valori RGB al di sopra di un limite soglia che dopo varie prove è stato impostato a 50.
Una volta trovato un pixel con tali caratteristiche, si chiama `OrthoPhoto::HoleChecking()` che identifica tutti i pixel contigui con tali proprietà, localizzando così non un macro-buco, bensì una macro-placca. In maniera del tutto analoga si identificano anche i bordi della stessa, utilizzando financo la stesse strutture dati.

4.9 Eliminazione delle placche

Le operazioni per la correzione delle placche avvengono, anche in questo caso, grazie alla procedura `OrthoPhoto::HoleSetting()` non prima, comunque, di averne individuato i bordi e calcolato un colore medio, analogamente a quanto accade per i macro-buchi.
La procedura utilizzata è la stessa dei pixel-buco, in quanto in tal caso i pixel-placca vengono trattati come se lo fossero, dato che i valori delle loro componenti RGB andranno ad essere rimpiazziati con altri valori per niente correlati ai primi.

4.10 Acquisizione delle coordinate dai file 'lab'

Nei file etichetta [cfr. paragrafo 3.4.1] sono presenti i dati per la localizzazione delle immagini nel mosaico digitale, come longitudine e latitudine in radianti, e le dimensioni dei lati dell'immagine in pixel. Informazioni, queste, che ricoprono un ruolo fondamentale poiché permettono di creare i file grafici della dimensione corretta e di localizzarli esattamente dove devono stare.
Per tale motivo, viene creato un file di testo con estensione 'pos' che accompagna il corrispondente file immagine 'bmp'. Alla fine della procedura di importazione, per ognuno dei file 'img', nel database dell'adattatore, che stiamo andando a creare, esisterà una coppia di file composta da un 'bmp' e un 'pos'.

Spiegazione dell'algoritmo

L'acquisizione delle coordinate e delle dimensioni avviene grazie all'uso della funzione `OrthoPhoto_Nasa::GetCoord()`, richiamata da `Filesystem_Package::UserProcessFile()`.
La prima operazione da fare è aprire in lettura il file etichetta 'lab' da cui, a questo punto, possone essere prelevate e memorizzate, in variabili membro, i valori numerici di longitudine e latitudine, indicanti i punti più estremi di copertura areale, del file nelle quattro direzioni cardinali: Nord, Sud, Ovest, Est. Inoltre vengono acquisite le dimensioni dei lati.
L'ultimo passaggio avviene in `Filesystem_Package::UserProcessFile()` che a questo punto può aprire un file di testo con estensione 'pos', scriverci le informazioni e crearlo fisicamente.
Il file segue un ordine rigido per scrivere i dati. In particolare viene salvata, come prima, la coppia di valori delle dimensioni dei lati dell'immagine e poi le coppie di coordinate dei vari angoli in senso anti-orario, a partire da quello in basso a sinistra, ovvero secondo l'ordine che segue: Sud-Ovest, Sud-Est, Nord-Est, Nord-Ovest.

Capitolo 5

Classi dell'importatore

Per questo primo applicativo di importazione delle immagini, è stato necessario creare varie classi (oggetti C++) che interagissero tra loro con l'unico scopo di ottenere un database di immagini di tipo 'bmp' da cui poi creare la collezione di immagini con cui generare, in seguito, il PID [cfr. capitolo 2].
Nel seguito, per non rendere troppo pesante la stesura del lavoro, nonché la lettura dello stesso, ho preferito non riportare il codice di ogni singola classe nella sua interezza, rinviando la completezza dell'analisi alla visione del codice sorgente del programma.
Vengono qui proposti, però, almeno i passaggi chiave delle routine in termini logici, senza perdere di vista il filo generale del discorso.

5.1 La classe ultratools

Questa contiene il programma principale `main()` e ricopre il ruolo di interfaccia primaria per l'utente. Qui vengono dichiarate, e se necessario istanziate, tutte le classi di base, utili all'importazione e non solo. Non facendo parte del progetto di tesi vero e proprio, in quanto precedentemente realizzata per poter usufruire di altri applicativi utili alla Fondazione Ultramundum, tale classe è stata solo lievemente aggiornata con le dichiarazioni di alcune delle nuove classi da me realizzate.

5.2 La classe OrthoPhoto_Package

La classe 'OrthoPhoto_Package' è stata creata per gestire l'input dalla riga di comando e, quindi, per dare il via all'esecuzione del programma. Le funzioni di una certa importanza sono essenzialmente due:

```
bool bProcessCommandLine(   const int argc,
                            char* argv[]);

void SpecUtilityInfo(       const char u8Select,
                            const bool bDetails);
```

bProcessCommandLine()

In base al valore di argc e alle stringhe contenute in argv[ ] decide come operare.
Restituisce il valore TRUE se il comando in input viene processato in modo corretto, FALSE altrimenti.
Per attivare la classe OrthoPhoto_Package sulla riga di comando deve apparire la sequenza "`ultatools OP [option]`", dove *option* identifica una specifica procedura e al momento può assumere quattro valori:

- ? per avere una rapida guida in linea;
- N per importare le immagini NASA e creare il database dell'adattatore con cui creare la collezione di immagini da usare per generare il PID;
- O per creare i file con le indicazioni dei valori di latitudine e longitudine dei file immagine della NASA, necessarie per il completamento del database;
- P per adattare le immagini del database al sistema UltraPort.

SpecUtilityInfo()

Sostanzialmente è un aiuto in linea: mostra su schermo le informazioni necessarie per poter inserire la giusta sequenza di comando. É richiamata da `bProcessCommandLine()` in caso di sequenza di input non corretta.
Il valore u8Select identifica la procedura che si vuole attivare, mentre bDetails indica se mostrare la descrizione particolareggiata degli aiuti.

5.3 La classe OrthoPhoto

Questa è una classe astratta che si occupa di gestire le foto ortografiche. La maggior parte dei metodi servono per la creazione, la manipolazione e la gestione delle immagini. Contiene due funzioni virtuali pure e questo vuol dire che viene usata come base per altre classi da essa derivate.
Le funzioni virtuali pure sono due:

```
virtual bool GetCoord(   const char* pszFilename) = NULL;

virtual bool GetData(    const char* pszFilename,
                         const char* pszFileJpeg) = NULL;
```

Ecco le altre importanti routine che gestiscono ogni singola immagine:

```
void OrthoPhotoInitialize(     const unsigned int unDx,
                               const unsigned int unDy);

void OrthoPhotoTestOut(        const char* pszFileName);

void StartOrthoPhotoTestOut(   const enum eImageFileType eType,
                               const unsigned int unDx,
                               const unsigned int unDy);

void OrthoPhotoTestPixelSet(   const unsigned int unX,
                               const unsigned int unY,
                               const unsigned char unRGBRed,
                               const unsigned char unRGBGreen,
                               const unsigned char unRGBBlue);

void EndOrthophotoTestOut(     const char* pszFileName);

void OrthoPhotoFromJpg(        const char* pszFileName);

void HoleChecking(             const unsigned long u32Index,
                               const unsigned int u16Type;)

void HoleSetting(              const unsigned long u32Index,
                               const unsigned int u16Type,
                               const unsigned int u16Option);
```

OrthoPhotoInitialize()

Inizializza il valore di gran parte delle variabili membro della classe OrthoPhoto.
I parametri unDx e unDy sono rispettivamente la dimensione orizzontale e quella verticale in pixel dell'intera immagine.

OrthoPhotoTestOut()

Questo metodo dà il via alla creazione dell'immagine. Il suo funzionamento è molto lineare, in quanto, tramite la chiamata ad altre funzioni, imposta il file grafico, ne setta i pixel e lo scrive su disco.
La stringa pszFileName corrisponde al nome del file immagine che si andrà a scrivere.

StartOrthoPhotoTestOut()

Imposta il tipo e le dimensione del file grafico da creare.
Il parametro eType è una costante enumerativa definita in tale classe che può assumere, a partire dal default UM_IMG_FTYPE_NONE, i valori UM_IMG_FTYPE_BMP o UM_IMG_FTYPE_JPEG in caso il file sia di tipo 'bmp' o 'jpg'. Inoltre viene istanziata un'area di memoria in grado di contenere tutti i dati necessari per la realizzazione di un'immagine della grandezza, in numero di pixel, di unDx × unDy, ovvero le dimensioni dei lati.

OrthoPhotoTestPixelSet()

Imposta, pixel per pixel, tutti i valori delle tre componenti cromatiche RGB.
I tre parametri unRGBRed, unRGBGreen e unRGBBlue, sono i valori dei canali RGB, utilizzati per settare le omologhe componenti dei pixel dell'immagine in memoria.

EndOrthophotoTestOut()

Semplicemente, riconosce il tipo di file per poi scriverlo su disco col nome indicato da pszFileName.

OrthoPhotoFromJpg()

Questa funzione permette di aprire un file 'jpg', il cui nome pszFileName è passato come parametro, e leggerne i valori delle componenti RGB, per poi memorizzarli nei vettori lineari m_pf64ImageBand1, m_pf64ImageBand2 e m_pf64ImageBand3, membri della classe, adibiti ad archivio temporaneo di tali preziose informazioni.

HoleChecking()

Procedura assai importante, in quanto è in grado di rintracciare, se presenti, i "buchi" nell'immagine e marchiarli secondo un precisa strategia. Per "buco" si intende un'area, dell'immagine, mancante o non raffigurata [cfr. sezione 4.2].
Il valore u32Index identifica l'indice del pixel che stiamo considerando, mentre u16Type dichiara esplicitamente il valore numerico identificativo del buco, a partire dal valore 1 a crescere.

HoleSetting()

Anch'essa è una procedura importante poiché prevede due funzionamenti diversi, ma ambo cruciali, identificati dal parametro u16Option. Il primo modo, infatti, prevede la copertura dei buchi secondo un colore calcolato proceduralmente; il secondo permette di identificare i pixel buoni che compongono il bordo di un macro-buco.
Anche in questo caso, u32Index indica il pixel che stiamo considerando, mentre u16Type è l'identificativo univoco del buco.

5.4 La classe OrthoPhoto_Nasa

Derivata dalla classe 'OrthoPhoto', contiene pochi ma significativi metodi specifici per la gestione e la manipolazione dei file immagine della NASA. Inoltre sono presenti le definizioni di quelle funzioni virtuali dichiarate nella classe da cui essa deriva.

```
bool GetCoord(      const char* pszFilename);

bool GetData(       const char* pszFilename,
                    const char* pszFileJpeg);

void MergeData(     OrthoPhoto *pCData1);

void FindHole();
```

GetCoord()

La funzione booleana in questione si occupa di aprire correttamente il file etichetta 'lab' del database della NASA identificato dal nominativo pszFilename e di acquisire le coordinate geografiche, in radianti, che servono a collocare il quadrante nella corretta posizione sull'intero mosaico della superficie lunare.
Restituisce il valore FALSE in caso il file venga regolarmente manipolato, TRUE altrimenti.

GetData()

Questa funzione membro consente di acquisire i valori delle tre componenti RGB di un'intera immagine. I due parametri richiesti sono nomi di file: pszFilename identifica l'etichetta 'lab' con la quale si potrà fare riferimento al file 'img' che contiene i valori delle cinque bande spettrometriche dell'immagine ad alta risoluzione della missione Clementine; pszFileJpeg, invece, individua il file 'jpg' con risoluzione minore al suo corrispondente file 'img'.
Restituisce il valore FALSE nel caso che i due file vengano regolarmente manipolati, TRUE altrimenti.

MergeData()

Il parametro pCData1 è un puntatore all'oggetto di tipo OrtoPhoto che contiene le informazioni dell'immagine 'jpg' corrispondente a quella 'img' che è in elaborazione. Tale routine consente di acquisire i colori dal 'jpg' e trasferirli all'immagine residente in memoria che contiene i dati delle bande del file 'img'.
Ad ogni pixel 'jpg' corrispondono 4 pixel 'img' in memoria che hanno, come base di colore, proprio il valore del pixel 'jpg'. Le bande del file 'img' servono a rendere meno omogeneo l'insieme di questi 4 pixel che altrimenti apparirebbe come un blocchetto 2×2 di pixel uguali.

FindHole()

Questa è una procedura che consente di avviare il riconoscimento di un pixel di buco, di zona grigia o di placca colorata. Una volta individuato, avvia la procedura per la ricerca degli altri pixel contigui con caratteristiche simili, marchiando tale agglomerato con un codice identificativo univoco.

5.5 La classe Filesystem_Package

Questa è una classe che consente di realizzare la ricerca di file e di chiamare le opportune funzioni, utili per l'importazione delle immagini, tramite le sue due routine più importanti:

```
bool UserProcessFile(   const char* pszDirectory,
                        const char* pszFile,
                        const int s16Option
                        const int s16Param1
                        const int s16Param2
                        const char* pszParam3);

bool ProcessFiles(      const char* pszDirectory,
                        const int s16Option,
                        const int s16Param1
                        const int s16Param2
                        const char* pszParam3);
```

UserProcessFile()

In questa procedura ci si accupa di manipolare il file pszFile nella directory pszDirectory. Gli interi s16Param1 e s16Param2, e la stringa pszParam3, sono i parametri acquisiti dalla riga di comando che permettono di manipolare il file secondo le proprie esigenze. In relazione al valore specificato da s16Option posso capire se inizierò le operazioni per la creazione di un file 'bmp' o di uno 'pos' per il database dell'adattatore.

ProcessFiles()

Questa procedura consente di ricercare i file all'interno della directory pszDirectory. Gli altri parametri servono solo per poter essere passati alla procedura per la manipolazione dei file trovati.

5.6 La classe Utility_Package

La classe qui in discussione ha quattro routine importanti.
In particolare contiene la funzione booleana

```
bool bProcessCommandLine(   const int argc,
                            char* argv[]);
```

e tre procedure:

```
void ProcessFiles(      const char* pszOutFileName,
                        const char* pszDirectory,
                        const int s16Option,
                        const int s16Param1,
                        const int s16Param2,
                        const char* pszParam3);

void UserProcessFile(   const char* pszOutFileName,
                        const char* pszDirectory,
                        const char* pszFile,
                        const int s16Option,
                        const int s16Param1,
                        const int s16Param2,
                        const char* pszParam3);

void SpecUtilityInfo(   const char u8Select,
                        const bool bDetails);
```

bProcessCommandLine()

In base al valore di argc e alle stringhe contenute in argv[ ] decide come operare.
Restituisce il valore TRUE se il comando in input viene processato in modo corretto, FALSE altrimenti.
Per attivare la classe OrthoPhoto_Package sulla riga di comando deve apparire la sequenza "`ultatools UT [option]`", dove *option* identifica una specifica procedura e al momento può assumere solo due valori:

- ? per avere una rapida guida in linea;
- B per creare un file batch utile per le procedure di importazione delle immagini per il database dell'adattatore;

ProcessFiles()

Questa procedura consente di ricercare i file all'interno della directory pszDirectory. Gli altri parametri servono solo per poter essere passati alla procedura per la manipolazione dei file trovati.

UserProcessFile()

Il valore di u16Option, che attualmente può assumere il solo valore 0, indica che la procedura si occuperà di assemblare la corretta riga di comando da scrivere nel file pszOutFileName, che viene creata concatenando una serie di stringhe e valori numerici.
Molto semplicemente, in una stringa temporanea di lavoro si concatenano la directory pszDirectory, il file ivi trovato pszFile e i parametri s16Param1, s16Param2 e s16Param3. La riga di comando è così pronta per essere scritta sul file pszOutFileName.

SpecUtilityInfo()

Sostanzialmente è un aiuto in linea: mostra su schermo le informazioni necessarie per poter inserire la giusta sequenza di comando. É richiamata da `bProcessCommandLine()` in caso di sequenza di input non corretta.
Il valore u8Select identifica la procedura che si vuole attivare, mentre bDetails indica se mostrare la descrizione particolareggiata degli aiuti.

Capitolo 6

Adattatore delle immagini

6.1 Panoramica

L'adattatore delle immagini è il secondo programma ideato per la realizzazione del PID lunare. Il suo ruolo è quello di creare i dati secondo lo standard in cui UltraPort è in grado di gestirli.
A partire dalla collezione di dati creata con l'importatore (è consigliabile far riferimento sempre alla figura 3.1), l'adattatore crea a sua volta un database in cui ogni immagine (che in questo contesto chiameremo anche *texture*) ha ben definite proprietà per quanto concerne la dimensione ed il tipo. Tale database, inoltre, è suddiviso in cartelle ognuna delle quali acquisisce, come vedremo, un preciso significato.
Un ruolo decisamente importante per la categorizzazione delle texture è ricoperto dal nome che idenitifica ogni file e cartella. Questo viene determinato secondo una logica di suddivisione delle aree dell'intera superficie lunare, basato sul riconoscimento del dettaglio grafico di risoluzione.
Dunque, una volta stabiliti i requisiti delle texture e delle directory, si è potuto iniziare a codificare le funzioni caratteristiche della classe *MoonPSector* che, appoggiandosi alla classe *OrtoPhoto_Package*, rappresenta l'adattatore, permettendo all'intera collezione di dati di conformarsi alle richieste di UltraPort.
La conoscenza di base della programmazione in C/C++, utile per la comprensione degli algoritmi, non richiede ulteriori capacità cognitive, se non una certa elasticità mentale per riuscire ad addentrarsi nella logica delle scelte informatiche effettuate.

6.2 Suddivisione logica della superficie

Nell'intero progetto del PID bisogna tenere conto della pesantezza, in termini di occupazione di memoria del processore, dei dati utili per la rappresentazione in tempo reale. Per la visualizzazione si tiene conto del punto in cui si trova l'osservatore, in gergo chiamato *avatar*.
La resa grafica, difatti, è determinata sostanzialmente dall'altitudine (rispetto al centro di massa del pianeta) a cui esso si trova, secondo una logica assolutamente semplice, quasi

banale: più ci si avvicina, maggiore deve essere la risoluzione delle immagini visibili. In fondo, è inutile avere in memoria le texture con il massimo livello di dettaglio se l'osservatore si trova posizionato sulla Terra. D'altro canto, avere i piedi poggiati sul suolo lunare e non essere in grado di visualizzare in modo adeguato i particolari renderebbe l'esperienza poco immersiva.
Inoltre, esiste il concetto importante della visione periferica: tutto ciò che è attorno al punto in cui è focalizzato il nostro sguardo, risulterà meno dettagliato.
La suddivisione logica della superficie riflette i principi ora esposti e permette di creare gli effetti grafici adeguati senza sovraccaricare la memoria.

6.2.1 Livelli di dettaglio grafico

Il livello di risoluzione grafica, come detto, dipende dalla distanza a cui si trova l'osservatore rispetto al centro di massa della Luna, o del generico pianeta. Per chiarire il concetto, si farà ora un esempio.
Si consideri una qualunque persona che volge lo sguardo verso l'orizzonte mentre cammina. Ad un certo punto, in lontananza, compare la sagoma di una casa, oggetto delle nostre considerazioni grafiche: si consideri tale distanza come la distanza massima entro la quale si ha la percezione visiva della casa stessa. Ogni passo fatto in direzione del nostro oggetto, ci avvicina ad esso e ci permette, poco alla volta, di apprezzare sempre maggiori dettagli costruttivi della casa.
Se alla distanza massima percepiamo solo il generico colore delle pareti, man mano che ci avviciniamo, apprezziamo sempre maggiori particolari come porte, finestre, lucernai, ringhiere, grondaie, scalini, battiscopa, tappeti, fino a percepire la rugosità delle mura.
In natura, questo è un processo visivo graduale, che in grafica digitale viene reso con tanti livelli di dettaglio come se a partire da 1000 metri di distanza vedessi porte e finestre, da 800 i lucernai, da 700 ringhiere e grondaie, e così via: una visualizzazione non continua ma, in termini tecnici, discretizzata.
Se ad ogni nuovo oggetto percepito associamo un livello di dettaglio, abbiamo quello che in ambito di grafica digitale si chiama *risoluzione discretizzata del dettaglio grafico*. A questo punto, ad ogni livello si potrebbe associare anche un identificativo numerico a partire da quello meno dettagliato in avanti. Nel nostro esempio risulterebbe:

livello 0 , tra la distanza massima di visualizzazione e 1000m: generico colore delle pareti;

livello 1 , tra 1000m e 800m: porte e finestre;

livello 2 , tra 800m e 700m: lucernai;

... ;

livello MAX , rugosità delle pareti.

Si sottolinea che i valori riportati per l'esempio sono a puro scopo dimostrativo e non hanno alcuna connotazione scientifica.

6.2.2 Visione periferica

Supponiamo, ora, di essere a 2 metri dalla casa (pensiamolo come livello massimo) e di avere gli occhi puntati sulla porta di casa. Eventuali ringhiere, battiscopa, tappeti, finestre e quant'altro non capiti nel nostro cono visivo, risulta avere un livello di dettaglio di molto inferiore a quello che noi apprezziamo e che è al centro della nostra attenzione. Si ha, pertanto, solo una percezione di ciò che ci circonda, senza poterne apprezzare i dettagli.

6.2.3 Realizzazione pratica

La commistione fra i due concetti sopra riportati, ci consente di spiegare in che modo venga realizzata la suddivisione della superficie in argomento.
Innanzitutto, torniamo allo scenario lunare. Le immagini del database formano l'intera superficie della crosta ed è possibile avere diversi livelli di dettaglio e di particolari. In modo analogo a quanto accade per la casa dell'esempio precedentemente descritto, anche per la Luna esiste una distanza massima oltre la quale scompare dalla nostra vista. Teoricamente, al livello 0 la Luna potrebbe essere rappresentata da una semplice sfera di colore piuttosto uniforme senza la possibilità di cogliere alcun particolare.
Per poter realizzare l'effetto di visione periferica si è pensato di realizzare una scomposizione in settori in relazione alla distanza dal centro di massa. Il concetto è che a partire dal livello 0, che rappresenta l'intera superficie, più ci si avvicina e più l'immagine viene spezzettata, in modo che l'osservatore circoscriva in maniera più specifica l'area che andrà a visualizzare, rendendo al massimo dettaglio solo quella parte che realmente lo interessa. Immaginiamo di essere su un'astronave nello spazio e di voler allunare in un preciso punto, come indicato in figura 6.1.

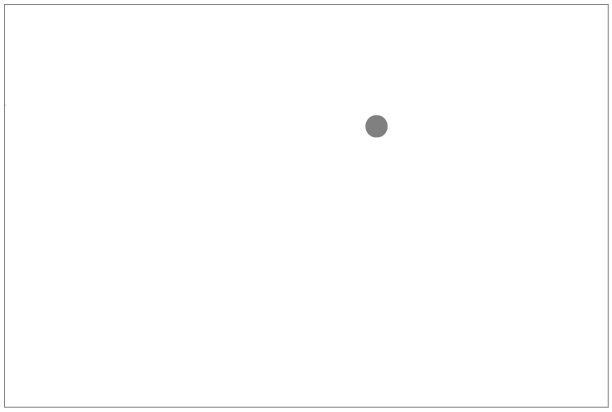

Figura 6.1. Ipotetico punto di allunaggio sulla rappresentazione piana della superficie lunare

Mano a mano che ci si avvicina al suolo lunare, aumentano i livelli di dettaglio e la superficie viene suddivisa in più parti. Il livello 0 viene suddiviso in due parti sull'asse verticale,

così da avere gli emisferi nord e sud, ognuno di livello 1. Ognuna di queste parti viene suddivisa in altre tre in orizzontale: si ottengono 6 pezzi di livello 2. Da questo momento, ogni pezzo verrà suddiviso in quattro parti (due in verticale e due in orizzontale) all'aumentare del livello di dettaglio. La figura 6.2 può aiutare a comprendere meglio il procedimento.

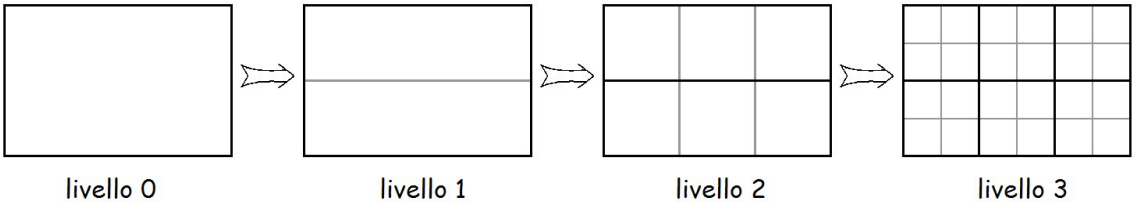

Figura 6.2. Progressiva suddivisione della superficie in relazione al livello di dettaglio grafico

Dunque, si può intuire in che modo venga individuato il punto d'allunaggio. Tramite una serie di confronti, si arriva a circoscrivere la zona: più si scende e più si spezzetta l'immagine di livello 0. Se si suppone di avere a disposizione delle immagini con livello massimo di dettaglio pari a 4, possiamo vedere un'esemplificazione dell'individuazione della zona nella figura 6.3, in cui si intuiscono le progressive suddivisioni, dal livello minimo fino a quello massimo, delle sole zone in cui cade il punto.

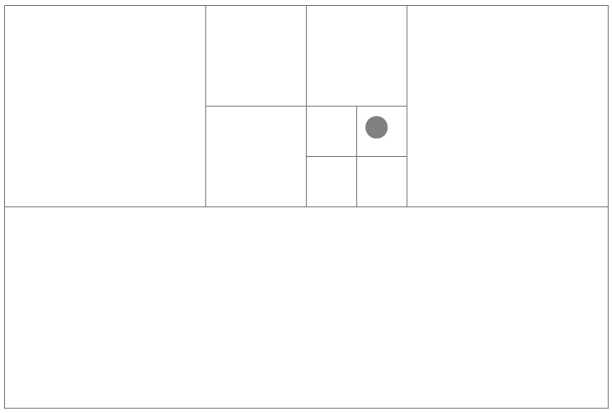

Figura 6.3. Individuazione del punto di allunaggio tramite successive suddivisioni della superficie

6.3 Esigenze di UltraPort

Il sistema UltraPort [cfr. paragrafo 1.1.3] permette di visualizzare su schermo l'oggetto 3D che nel nostro caso è la Luna e che, digitalmente parlando, è una sfera con le texture della superficie adeguatamente sistemate sopra.
Le texture devono avere precise caratteristiche in termini di dimensioni e di tipo. Inoltre

la loro nomenclatura deve rispettare delle direttive da utilizzare anche per le directory dove esse sono salvate.

6.3.1 Proprietà delle texture

Le texture sono dei file grafici quadrati di tipo 'bmp' della dimensione di 256 pixel per lato. Ogni file rappresenta un settore planetario di superficie, più o meno grande, chiamato *Planet Sector*, o più semplicemente PS. L'area rappresentata dipende dal livello di dettaglio che è possibile raggiungere, in dipendenza delle immagini ortografiche a disposizione.

6.3.2 Identificazione dei Planet Sector

Per ognuno dei livelli di dettaglio vengono creati tanti settori planetari quanti ne occorrono per comporre l'intera superficie, secondo la tabella 6.1. Ognuno di essi, inoltre, viene identificato tramite un nome univoco nella forma "`a_x_y`" in cui:

- `a` indica il livello di dettaglio con una lettera maiuscola dell'alfabeto inglese: al livello 0 corrisponde la 'A', al livello 1 la 'B', al 2 la 'C' e così via;
- `x` indica l'indice orizzontale di posizionamento del Planet Sector ed è codificato in base esadecimale;
- `y` indica, invece, l'indice verticale di posizionamento del Planet Sector, anch'esso in esadecimale.

Identificatore del livello	Numero dei PS in orizzontale	Numero dei PS in verticale	Numero totale dei PS
0 - A	1	1	1
1 - B	1	2	2
2 - C	3	2	6
3 - D	6	4	24
4 - E	12	8	96
5 - F	24	16	384
6 - G	48	32	1 536
7 - H	96	64	6 144
8 - I	192	128	24 576
9 - J	384	256	98 304
10 - K	768	512	393 216

Tabella 6.1. Numero di PS in relazione al livello di detttaglio

Facendo riferimento all'esempio del punto di allunaggio ed alla figura 6.4, il Planet Sector di massimo dettaglio verrebbe identificato come `E_7_5`.

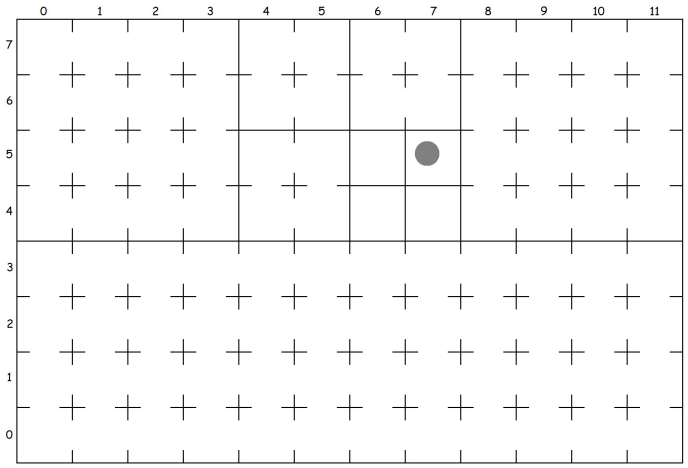

Figura 6.4. Suddivisione e indicizzazione del Planet Sector di allunaggio

6.3.3 Struttura del database del PID

Il database del PID è formato di tante directory annidiate che contengono i file afferenti ai vari settori planetari.

Caratteristiche dei file dei Planet Sector

Per la corretta rappresentazione, per ogni PS sono necessari due file che posseggono lo stesso identificativo e differiscono solo per l'estensione del nome.
Il primo è un file grafico di tipo 'bmp' e contiene l'immagine vera e propria che si andrà a disegnare sullo schermo.
Il secondo è un file testo di tipo 'txt' in cui sono scritte le coordinate dei quattro angoli in termini di longitudine e latitudine. Contrariamente a quanto accade di norma nelle più comuni cartografie, in cui la latitudine varia nell'intervallo $(-\pi/2, +\pi/2)$, in UltraPort i valori sono compresi in $(0,\pi)$, che corrispondono rispettivamente agli estremi a sud e a nord.

Organizzazione delle directory

Il database del PID è una directory strutturata secondo una precisa idea di fondo.
A partire dal primo settore planetario, ogni PS può essere suddiviso in altri più piccoli fino al raggiungimento del massimo livello di dettaglio. Ecco che, in linea teorica, un PS è una scatola che ne contiene altre minori.
Il passaggio alla memorizzazione tramite directory è breve. Ogni directory (che rappresenta un PS "padre") contiene i file dei PS suoi "figli" ed eventuali cartelle se questi possono essere scomposti in altri settori.
Per capire meglio, la directory `A_0_0` contiene i file (txt e bmp) `B_0_0` e `B_0_1`, ma anche le cartelle, nominate allo stesso modo, in quanto i due PS stessi possono essere suddivisi

in altri. Parimenti, B_0_0 conterrà (sia come file che come cartelle) C_0_0, C_1_0 e C_2_0, mentre B_0_1 avrà C_0_1, C_1_1 e C_2_1. In sintesi, a partire dalla prima cartella, si ha una nidificazione di directory in maniera che il concetto di suddivisione plurima e ricorsiva venga mantenuto anche nella struttura di memorizzazione dei dati.
Tornando, dunque, all'esempio del punto di allunaggio, il percorso completo del file 'bmp', corrispondente al PS di nostro interesse, sarebbe "A_0_0\B_0_1\C_1_1\D_3_2\E_7_5.bmp".

6.4 Procedura di adattamento

La procedura è unica ma, volendo, vi si possono riconoscere due fasi: una di preparazione e quella dell'effettivo adattamento delle immagini. La prima crea tutta la struttura delle cartelle scrivendovi anche i file di testo che contengono le coordinate geografiche. La seconda, invece, permette di creare i settori planetari a partire dal database di immagini e di scriverli nelle directory precedentemente create.
L'intera procedura prende il via tramite l'attivazione della classe OrthoPhoto_Package con il comando "ultratools OP P"[cfr. sezione 3.7].

6.4.1 Generico funzionamento

Comandi di avvio

L'adattamento delle immagini si avvia inserendo la riga di comando

```
ultratools OP P <dir1> <par1> <par2>
```

in cui i parametri hanno il seguente significato:

<**dir1**> è la directory del database dove andare a cercare i file di partenza da trasformare in settori planetari.

<**par1**> è una stringa di caratteri che identifica il nome del pianeta che si vuole adattare: nel nostro caso è "Moon".

<**par2**> è un intero ed indica il numero massimo di livello di dettaglio che si vuole raggiungere. Il valore da immettere dipende dalla conoscenza che si ha delle caratteristiche tecniche delle immagini del proprio database: allora è possibile mettere un valore con cognizione di causa. Nel caso si voglia far calcolare il valore di dettaglio grafico in modo automatico (scelta vivamente consigliata) è sufficiente immettere il valore 0, impostato per default.

Una riga di comando corretta risulterebbe come la seguente:

```
ultratools OP P C:\ClementineMission\MoonDatabase Moon 0
```

Dopo tale comando, la procedura si avvia con la determinazione in automatico del livello di dettaglio, la creazione delle cartelle e, infine, dei Planet Sector.

Funzionamento

La sequenza delle operazioni necessarie all'adattamento, risulta, rispetto a quella di importazione, sensibilmente più complessa, in quanto sono richieste più funzioni per la corretta

gestione delle texture dei settori planetari.
La funzione `OrthoPhoto_Package::bProcessCommandLine()`, dopo aver controllato la correttezza dei parametri in ingresso, avvia l'intera procedura che, sotto l'aspetto logico, possiamo suddividere in varie fasi:

1. impostazione della directory di output e della classe base per permettere la chiamata delle adeguate routine;
2. calcolo, se necessario, del massimo livello di dettaglio grafico per mezzo della funzione `MoonPSector::CalculateMaxPSLvl()`;
3. creazione delle directory, e dei relativi file di coordinate geografiche, tramite la procedura `MoonPSector::CreatePSDirs()`;
4. creazione e scrittura dei PS di livello massimo con `MoonPSector::ManagePSLevel()`;
5. accorpamento dei PS figli creando un PS padre, a partire dal massimo livello fino al raggiungimento di quello minimo, con la procedura `MoonPSector::MergePSsManager()`.

È bene chiarire ora, e in modo non troppo particolareggiato, le operazioni ai punti 4 e 5 del precedente elenco.
In effetti, il programma in un primo momento si occupa di creare i file immagine dei PS al massimo livello di dettaglio disponibile che nel nostro progetto lunare arriva ad essere 10 per alcuni settori e 9 per altri, in dipendenza del fatto che si trovino nella fascia centrale della Luna o in zone prossime ai poli.
Una volta creati tali PS di livello massimo, si rende necessario comporre, a partire da questi, tutti i PS di livello inferiore. Così, per esempio, tutti i PS di livello 8, presi a gruppi di quattro come da specifica suddivisione logica della superficie, andranno a formare i corrispettivi PS padri di livello 7. Quelli di livello 7 si uniranno nei PS di livello 6 e così fino ad arrivare all'ultimo PS di livello 0 che rappresenta l'intera crosta lunare.

Capitolo 7

Algoritmi dell'adattatore

7.1 Vista d'insieme

Prima di procedere con l'adattamento delle immagini vero e proprio, è necessaria una fase di preparazione. Per sommi capi, dunque, è possibile suddividere in due parti, almeno a livello logico, l'intera procedura. Inoltre, anche leggendo quanto riportato in sezione 6.4, ognuna di queste, può essere scissa in due sotto-fasi.
Quanto detto potrebbe essere schematizzato come segue:

Fase di preparazione :

- calcolo, se necessario, del livello massimo di dettaglio [cfr. sezione 7.4];
- creazione delle cartelle e dei file di coordinate [cfr. sezione 7.5].

Fase di adattamento :

- creazione dei PS di massimo dettaglio [cfr. sezione 7.7];
- accorpamento di tutti i settori planetari [cfr. sezione 7.11].

7.2 Fase di preparazione

Questa prima fase consiste nel settare in maniera corretta le strutture elementari e la directory di base per il salvataggio di tutti i Planet Sector. In seguito vengono calcolati il livello massimo e minimo di dettaglio grafico delle immagini a disposizione. Infine, prima di iniziare la fase di adattamento vero e proprio, si creano tutte le sotto-cartelle, corrispondenti ai vari PS, scrivendovi anche i file contenenti le coordinate geografiche.
Tutta questa prima fase risulta abbastanza semplice e lineare, nonché di facile comprensione.

7.3 Impostazione della directory del database del PID

Un primo importante passo della prima fase è l'impostazione e la creazione della directory di base dove poi verranno create tutte le sotto-cartelle e le immagini afferenti ai PS.

Per come impostato il programma, viene creata una cartella "`..\DataOut`" all'interno della directory dell'eseguibile del programma. In seguito, combinando il parametro della riga di comando in cui viene specificato il nome del pianeta [cfr. sezione 6.4], si crea la cartella del database che nel nostro caso risulta "`..\DataOut\Moon\MoonTextures`".
In seguito viene creato il file 'WarningFile.txt' in cui vengono scritti gli avvisi relativi alle fasi di elaborazione raggiunte, ma anche eventuali messaggi di errore durante l'esecuzione del programma.
Infine, si istanzia un settore planetario di base, una sorta di PS fittizio, di livello 0-A che servirà per richiamare tutte le routine utili all'intera procedura di adattamento.

7.4 Calcolo del livello di dettaglio

La prima routine importante, che opzionalmente viene richiamata, è quella che permette di calcolare i livelli di massimo e minimo della risoluzione grafica. La possibilità di attivazione dipende dall'ultimo parametro della riga di comando [cfr. sezione 6.4]: nel caso valga 0, viene chiamata la funzione `MoonPSector::CalculateMaxPSLvl()`.
Questa passa in rassegna tutti i file del database dell'adattatore e confronta le dimensioni dei pixel delle immagini con quelle dei pixel dei Planet Sector.

7.4.1 Caratteristiche di un pixel

Sebbene parzialmente già definito in precedenza [cfr. paragrafo 3.5.2], il pixel è da considerarsi come area e non come punto geometrico. Questo implica che ognuno di essi possegga delle coordinate di massimo e di minimo che lo definiscono sia in larghezza che in altezza.
I PS sono delle texture dalla dimensione di 256 pixel per lato [cfr. sezione 6.3], ma devono rappresentare, in relazione al livello di dettaglio grafico, aree di diverse dimensioni.
Per capire, si considerino i settori planetari A_0_0 e B_0_0, che rispettivamente rappresentano l'intera superficie del globo e la sua metà inferiore, e si focalizzi l'attenzione sulle dimensioni verticali. Nel primo caso, i 256 pixel del singolo PS devono coprire un'altezza che, in radianti, assume valori compresi tra 0 e π, mentre nel secondo tra 0 e $\pi/2$. In quest'ultimo caso, la texture ricopre meno superficie poiché ogni livello viene diviso in PS equidimensionali, ognuno dei quali ha pixel di medesima larghezza e altezza. Conseguentemente, ognuno dei 256 pixel rappresenterà una porzione di superficie con un'estensione verticale minore.
I file immagine del database, invece, hanno sia dimensioni in pixel sia estensioni superficiali diverse. Si può dedurre che i pixel di una determinata immagine, oltre a ricoprire una porzione d'area diversa, hanno dimensioni differenti da quelli di un'altra.
Le dimensioni orizzontale e verticale di un pixel si trovano con una semplice divisione:

$$\Delta X_{pix} = \frac{\Delta X_{img}}{N_{Pix_X}} \quad ; \quad \Delta Y_{pix} = \frac{\Delta Y_{img}}{N_{Pix_Y}}$$

dove ΔX_{pix}, ΔY_{pix}, ΔX_{img}, ΔY_{img} sono le estensioni in orizzontale ed in verticale, in

radianti, dei pixel e dell'immagine, mentre N_{Pix_X} e N_{Pix_Y} sono il numero di pixel sui rispettivi assi.

Spiegazione dell'algoritmo

La funzione `MoonPSector::CalculateMaxPSLvl()` attiva `MoonPSector::ProcessFile()` che permette di scandire tutti i file presenti all'interno della directory passata come parametro.
Non appena viene trovato un file 'pos', viene chiamata `MoonPSector::UserProcessFile()`, che come vedremo si ritrova ad avere più ruoli importanti, in quanto fulcro di varie operazioni attivate, di volta in volta, secondo le esigenze specifiche. In questo frangente permette di avviare la procedura `MoonPSector::SetMaxLevelByPixel()`, dove avviene il confronto tra dimensioni dei vari tipi di pixel.
A partire dallo 0, vengono istanziati i PS per ogni livello fino a che si trova che le dimensioni del pixel del PS sono ambo minori di quelle del pixel dell'immagine. Contestualmente, il valore del livello trovato viene confrontato con il massimo ed il minimo fino a quel momento rilevati.
Alla fine della scansione dei file di tutte le cartelle, si avranno i valori di massimo e minimo assoluti, per quel che concerne il livello di dettaglio grafico.

7.4.2 Esempio di calcolo dei livelli di dettaglio

Il file 'ui87s030.pos' mi dice che il corrispettivo file immagine di 1906×1819 pixel ha un'estensione che varia nell'intervallo $(0,\ 1{,}049\,399)$ in orizzontale e in $(0,\ 0{,}104\,719)$ in verticale: risulta che ogni suo pixel ha dimensioni in radianti pari a

$$\Delta X_{pix_{img}} = \frac{1{,}049\,399}{1\,906} = 0{,}550\,576 \times 10^{-3} \quad ; \quad \Delta Y_{pix_{img}} = \frac{0{,}104\,719}{1\,819} = 0{,}057\,569 \times 10^{-3}$$

Il programma istanzia un Planet Sector di livello 0-A: se l'estensione in ambo le dimensioni del suo pixel non è minore di quanto appena calcolato, istanzia un PS di livello successivo. Tale procedimento si interrompe non appena si verifica quanto detto: si confronta, dunque, il valore del livello dell'ultimo PS istanziato con quelli di massimo e minimo, impostandoli se necessario.
Facendo qualche rapido calcolo si può verificare che per il livello 8-I si ottengono

$$\Delta X_{pix_{PS(8)}} = \frac{2\,\pi}{192 \cdot 256} = 0{,}127\,831 \times 10^{-3} \quad ; \quad \Delta Y_{pix_{PS(8)}} = \frac{\pi}{128 \cdot 256} = 0{,}095\,873 \times 10^{-3}$$

mentre per il livello 9-J

$$\Delta X_{pix_{PS(9)}} = \frac{2\,\pi}{384 \cdot 256} = 0{,}063\,915 \times 10^{-3} \quad ; \quad \Delta Y_{pix_{PS(9)}} = \frac{\pi}{256 \cdot 256} = 0{,}047\,936 \times 10^{-3}$$

Quindi, da un veloce confronto, risulta che il livello massimo di dettaglio è il 9-J in quanto

$$\Delta X_{pix_{PS(9)}} \leq \Delta X_{pix_{img}} \quad \text{e} \quad \Delta Y_{pix_{PS(9)}} \leq \Delta Y_{pix_{img}}$$

cosa non vera per il livello 8-I.
In tal caso, il valore 9 viene salvato sia come valore massimo assoluto che come valore minimo assoluto.
Completando la scansione delle cartelle, però, risulteranno dei file per cui è possibile raggiungere livello 10-K. In tal caso, viene modificato solo il livello massimo assoluto.

7.5 Creazione delle cartelle dei PS

Stabilito il livello massimo di dettaglio, si prosegue con la creazione di tutte le sotto-cartelle per mezzo della procedura `MoonPSector::CreatePSDirs()`, chiamata all'interno di `OrthoPhoto_Package::bProcessCommandLine()` tramite il PS fittizio di livello 0-A.
Il suo è un funzionamento di tipo recursivo in quanto in ognuna delle cartelle che rappresentano un PS, crea le sotto-cartelle relative (fino ad arrivare ad un livello limite pari al massimo livello di dettaglio meno uno) e, contestualmente, i rispettivi file con le coordinate dei PS rappresentati.

Spiegazione dell'algoritmo

La procedura inizia dal livello 0-A fino ad arrivare all'ultimo possibile, ovvero quello limite, creando la directory "`A_0_0`" in quella di base "`..\DataOut\Moon\MoonTextures`", avendo perciò la prima cartella "`..\DataOut\Moon\MoonTextures\A_0_0`".
A questo punto entra in gioco la recursività dell'algoritmo. Una volta impostata tale directory padre, infatti, si procede alla creazione di tutte le sotto-cartelle figlie.
Con un ciclo tipicamente matriciale che prende in esame tutti i sotto-PS del settore planetario corrente, si settano gli indici corretti che, di volta in volta, andranno a costituire il nome della directory corrispondente al PS da rappresentare. Per ogni PS così identificato, si crea la cartella e si richiama la medesima procedura che inizierà un suo ciclo interno per la creazione dei propri sotto-settori (e sotto-cartelle).
Per ogni cartella generata, si avvierà una procedura per la creazione delle prorie sotto-cartelle fino al raggiungimento del livello limite.

7.5.1 Esempio di creazione delle cartelle

Si supponga di avere il livello massimo di dettaglio 3-D (e quindi un livello limite per le cartelle pari a 2-C) e di aver appena generato la cartella "`A_0_0`".
Il PS A_0_0 contiene i due sotto-settori B_0_0 e B_0_1. L'algoritmo setta gli indici orizzontale e verticale ambo pari a 0. Viene generata, pertanto, la prima sotto-cartella "`A_0_0\B_0_0`".
Poichè non si è raggiunto il livello di dettaglio limite, si prende in considerazione il PS B_0_0 che contiene i tre sotto-settori C_0_0, C_1_0 e C_2_0. Viene ora generata la prima sotto-cartella "`..\B_0_0\C_0_0`".
Il livello 2-C è quello di limite per le directory: l'algoritmo non crea nuove sotto-cartelle

che non avrebbero senso di esistere in quanto una cartella di livello 3-D dovrebbe contenere file di livello 4-E che nel nostro esempio non è possibile avere. Ci si limita, dunque, a creare i file inerenti al massimo livello di dettaglio grafico disponibile 'D_0_0.txt', 'D_1_0.txt', 'D_0_1.txt' e 'D_1_1.txt' all'interno della cartella "`..\B_0_0\C_0_0`". Solo ora verrà creato il file 'C_0_0.txt' nella cartella "`A_0_0\B_0_0`".
Completata la prima sotto-directory di B_0_0, si passa alla seconda, ovvero C_1_0 (che non conterrà alcuna cartella ma solo i file 'D_2_0.txt', 'D_3_0.txt', 'D_2_1.txt' e 'D_3_1.txt') generando anche 'C_1_0.txt' , e poi alla terza C_2_0 (con i file 'D_4_0.txt', 'D_5_0.txt', 'D_4_1.txt' e 'D_5_1.txt') e relativo 'C_2_0.txt'.
Completata, così, la prima sotto-cartella di A_0_0, si passa alla seconda, ovvero B_0_1, per la quale vale un discorso analogo.
I file di testo, dunque, vengono generati quando non è possibile creare altre sotto-cartelle, quasi ad indicare che le operazioni preparative, per quel dato sotto-settore, sono terminate. Si capisce, quindi, che l'ultimo file di testo ad essere scritto sarà 'A_0_0.txt'.
Alla fine dell'intera procedura, si ottiene un completo albero di directory nidificate il cui ultimo grado consiste in directory di livello 2-C contenenti file di livello 3-D, come mostrato in figura 7.1.

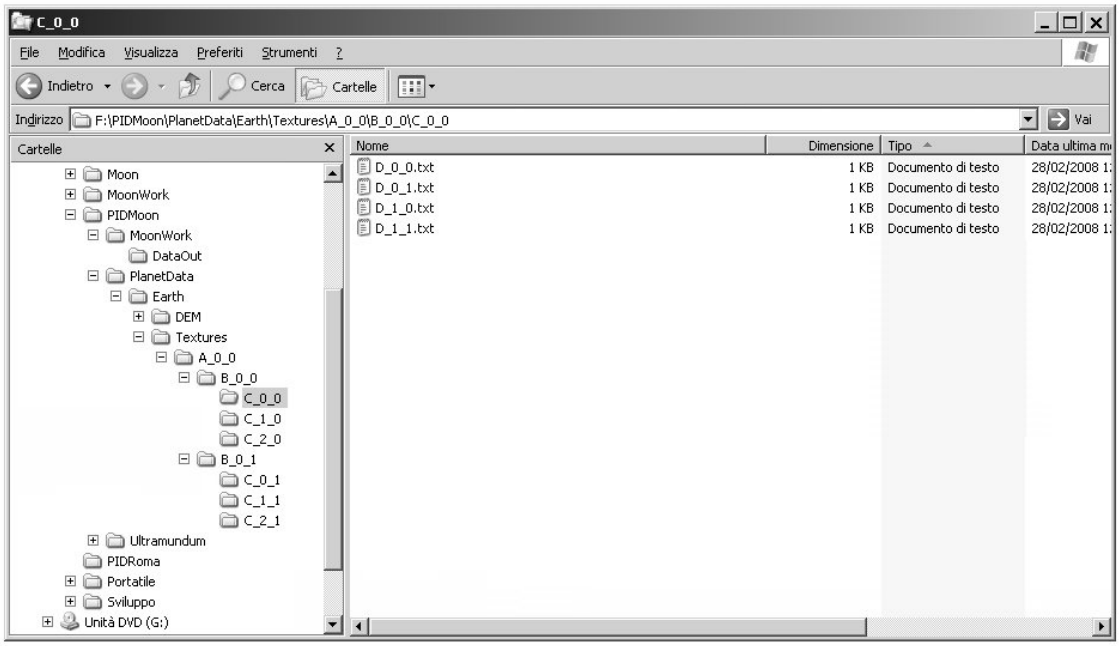

Figura 7.1. Struttura delle cartelle con livello massimo di dettaglio 3-D

7.6 Fase di adattamento

Questa seconda fase, che è quella effettiva di adattamento, richiede maggiore attenzione rispetto alla precedente poiché risulta molto articolata, in quanto composta da molteplici routine che interagendo tra loro permettono la realizzazione di quello che sarà il database del PID.
Anche questa, come la precedente, è possibile dividerla in due macro-fasi: la prima si occupa di creare i file immagine dei soli Planet Sector di maggior dettaglio grafico [cfr. sezione 7.7], mentre la seconda, a partire da quelli di base così generati, crea i PS di tutti gli altri livelli [cfr. sezione 7.11].

7.7 Creazione dei PS di base

Dall'interno di `OrthoPhoto_Package::bProcessCommandLine()`, a partire dal solito PS fittizio di livello 0-A, si chiama `MoonPSector::ManagePSLevel()`. Questa procedura consente di gestire un livello di dettaglio alla volta, a partire da quello massimo fino al minimo.
Dopo l'impostazione di alcune variabili, tramite `MoonPSector::bCheckDetailLevel()` (che si appoggia alle routine `MoonPSector::ProcessFile()`, `MoonPSector::UserProcessFile()` e `MoonPSector::SetMaxLevelByPixel()`) si controlla che per il determinato livello corrente esista almeno un file che raggiunga tale livello di dettaglio. Nel qual caso viene chiamata la procedura `MoonPSector::ManagePSector()`.
Quest'ultima permette di agire sui PS, uno alla volta, scandendoli come se fossero una matrice, prima lungo la direzione orizzontale e poi in verticale. Per ognuno di essi è necessario verificare, in primis, quali file del database lo includono anche solo parzialmente. Con tale operazione, a livello logico si vengono a sovrapporre la superficie della Luna, suddivisa in immagini dalla NASA, e quella del PID, formata di tanti settori planetari.
La ricerca della sovrapposizione parte con la procedura `MoonPSector::ProcessFile()`, ma si realizza tramite `MoonPSector::UserProcessFile()` e `MoonPSector::bCompareCoord()` che, però, non si limitano solo a tale obiettivo. Infatti, operando una serie di controlli, permettono di avviare gli algoritmi per l'acquisizione dei dati, ovvero i valori numerici delle componenti RGB, per la creazione della texture del PS in esame.
Una volta finito di controllare tutti i file per un determinato PS, in caso di esito positivo dei controlli e avendo così dei dati a disposizione, si procede prima a settare e poi a scrivere, in forma di file immagine di tipo 'bmp', il Planet Sector obiettivo delle nostre verifiche. Una volta completato l'intero ciclo dei PS e di tutti i livelli di dettaglio, si possiede una collezione di immagini base con la massima risoluzione grafica possibile. Il database del PID da noi creato parte da una serie di settori planetari di livello 9-J e 10-K, livelli di dettaglio differenti a causa delle diversità tra i file grafici di partenza della NASA.

7.7.1 File grafici buoni e cattivi

È doveroso porre l'attenzione su una questione: esiste la possibilità di creare file non buoni identificati con la particolare dicitura "_raw" posta tra il nome del file e l'estensione

'bmp': un ipotetico 'M_0_0_raw.bmp' sarebbe un file di un PS cattivo di livello 12-M, mentre 'M_0_0.bmp' sarebbe il corrispettivo buono.
La differenza tra i due tipi di file, risiede nel modo in cui sono colorati i suoi pixel, oltre che nel significato logico. Difatti, i file "raw.bmp" (d'ora in poi semplicemente 'raw') sono file grafici con qualche errore: precisamente, nel contesto del nostro adattatore, si vuole intendere pixel a cui non è associato alcun dato e colorati di nero con codifica (0,0,0). Nei PS buoni, e dunque nei file 'bmp', sebbene siano presenti pixel neri, essi vengono codificati con la terna (1,1,1).
Attenzione: questo tipo di codifica e di differenziazione tra file buoni e cattivi, vale solo ed esclusivamente per l'adattatore, e non per l'importatore.

7.8 Controllo sull'adeguatezza delle immagini del database in relazione al PS

All'interno di `MoonPSector::UserProcessFile()`, una volta verificato, per mezzo della funzione `MoonPSector::bCompareCoord()`, che un file si sovrappone ad un PS, si attuano una serie di controlli per capire se il livello grafico del file in analisi sia effettivamente compatibile con quello in elaborazione, soprattutto per evitare che nelle zone a livello 9-J, del nostro database del PID, vengano creati PS di livello 10-K.

Spiegazione dell'algoritmo

Una prima operazione compiuta è quella di prevedere quali sotto-settori potrebbero esserci nel PS in elaborazione e contare il corrispettivo numero di file grafici buoni e cattivi presenti nell'eventuale sotto-cartella.
Il primo controllo viene effettuato confrontando altezza e larghezza del pixel del PS con quello dell'immagine NASA: se almeno una, delle due realtive al PS, è maggiore della corrispettiva dell'immagine, si controlla la quantità dei file grafici. Se sono presenti tutti i sotto-settori che dovrebbero esserci o se non ne è presente alcuno, vuol dire che il file presente non apporterà alcun contributo al PS corrente, altrimenti sì.
Nel caso il primo controllo dia esito diverso (ovvero che ambo le dimensioni del pixel PS siano inferiori a quelle dell'immagine), si fa il controllo sulle dimensioni dei pixel del livello immediatamente inferiore, per verificare che non si scriva un PS di livello 10-K quando in realtà ce ne vorrebbe uno 9-J.
Per tutti i confronti i cui esiti indicano che un file apporta contributi per la creazione del PS, viene chiamata la procedura `MoonPSector::CreatePSTextures()`.

7.9 Calcolo dei valori RGB per i pixel della texture del PS

Grazie a `MoonPSector::CreatePSTextures()` è possibile effettuare tutte le necessarie operazioni per creare in memoria una struttura dati in grado di rappresentare la texture di un determinato Planet Sector.
Tutti i pixel dell'immagine che si sovrappongono anche solo parzialmente ad uno dei pixel

del PS, apportano un contributo per il computo finale delle componenti RGB del settore planetario.
I valori vengono memorizzati, chiamando la procedura `MoonPSector::AddPixelInstance()` e in via temporanea, in una struttura dati che rappresenta un PS di 256×256 pixel, ognuno dei quali in grado di memorizzare i tre valori RGB ad esso relativi.

7.9.1 Struttura dati per la composizione delle texture

Per comodità sia di interpretazione logica sia di implementazione, la struttura dati utile alla generazione delle texture del PS è stata realizzata in forma di matrice di 256×256 elementi di base, ognuno dei quali rappresenta un pixel.
Gli elementi di base sono delle variabili di tipo struttura come segue

```
struct m_SPSPixPropertyStruct{   float f32CoverageRate;
                                 unsigned char v3u8RGBValues[3];
                                 m_SPSPixPropertyStruct* PSSamePixNext; };
```

Il valore in virgola mobile f32CoverageRate indica la percentuale di copertura del pixel del file rispetto al PS e assume valori tra 0 e 1: più alto è il valore, maggiore è la superficie del pixel del PS coperta da quello dell'immagine.
Il vettore v3u8RGBValues, invece, serve per memorizzare i valori delle tre componenti cromatiche RGB del pixel medesimo.
Infine, è presente un puntatore ad un elemento dello stesso tipo della struttura per permettere di creare una lista concatenata lineare. Questo è necessario sicché ogni pixel della texture potrebbe ricevere contributi cromatici da più pixel delle immagini, non essendo, quest'utlimi, allineati alla griglia formata dai PS.
Per come è stato realizzato l'intero programma, al massimo livello di dettaglio un pixel dei PS risulta più piccolo in ambo le dimensioni di uno dell'immagine che, conseguentemente, si sovrapporrà a più pixel PS. A proposito, nell'esempio in figura 7.2, i pixel dell'immagine sono delimitati in nero e quelli del PS in grigio chiaro. Si fa notare anche che, alcuni pixel PS sono completamente ricoperti da quelli immagine, mentre altri, in maggioranza, lo sono solo parzialmente.

Spiegazione dell'algoritmo

Il primo passo compiuto è l'apertura in lettura del file 'bmp' dell'immagine, del database dell'adattatore, che si sovrappone al PS in elaborazione. A questo punto si scandisce l'immagine pixel per pixel, muovendosi prima sulle righe e poi sulle colonne.
Ogni pixel è necessario che venga localizzato sull'intera superficie: si calcolano, perciò, $ImgPixY_{min}$, $ImgPixY_{MAX}$, $ImgPixX_{min}$ e $ImgPixX_{MAX}$, ovvero le coordinate, sempre in radianti, di minimo e massimo relative agli assi verticale e orizzontale. Con queste informazioni è possibile calcolare i valori numerici degli indici verticali dei PS coperti dal pixel

$$PSYPos_{min} = \frac{ImgPixY_{min}}{\Delta Y_{img}} \quad ; \quad PSYPos_{MAX} = \frac{ImgPixY_{MAX}}{\Delta Y_{img}}$$

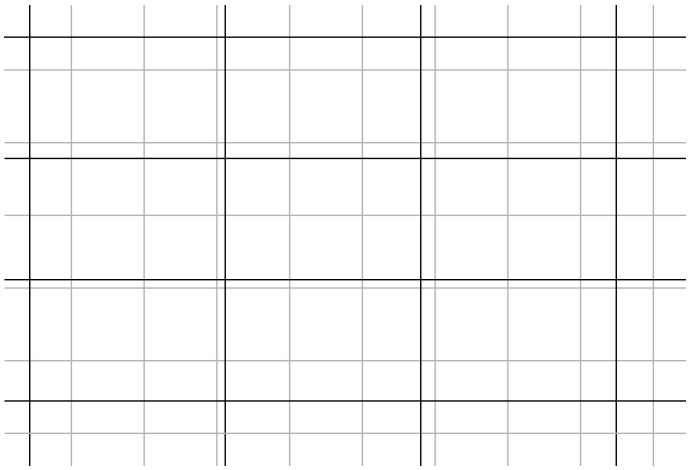

Figura 7.2. Corrispondenza ipotetica tra pixel dell'immagine NASA e pixel del PS

ed anche quelli orizzontali

$$PSXPos_{min} = \frac{ImgPixX_{min}}{\Delta X_{img}} \quad ; \quad PSXPos_{MAX} = \frac{ImgPixX_{MAX}}{\Delta X_{img}}$$

Gli indici, per definizione interi, così calcolati si riferiscono alle estensioni in verticale ed in orizzontale del pixel, mentre ΔX_{img} e ΔY_{img} sono le dimensioni dell'immagine del database.
In questo modo si capisce quali PS sono interessati dal pixel dell'immagine preso in considerazione: può accadere, infatti, che un pixel stia a cavallo tra più PS.
Ora, se la funzione di controllo `MoonPSector::bNasaPixIsInPS()` dà esito positivo, vuol dire che il pixel dell'immagine che è in elaborazione si relazioni con il PS per cui si vuole creare la texture. Può succedere, tuttavia, che solo parte di esso cada dentro il PS medesimo.
Si calcolano, dunque, ulteriori indici che permettono di stabilire esattamente quali pixel del PS siano coinvolti, in modo piuttosto analogo a quanto fatto precedentemente:

$$PSPixYPos_{min} = \frac{ImgPixY_{min} - PSYPos_{min} \cdot \Delta Y_{img}}{\Delta Y_{pix_{PS}}}$$

$$PSPixYPos_{MAX} = \frac{ImgPixY_{MAX} - PSYPos_{MAX} \cdot \Delta Y_{img}}{\Delta Y_{pix_{PS}}}$$

I valori interi $PSPixYPos_{min}$ e $PSPixYPos_{MAX}$ sono gli indici che permettono di identificare i pixel del PS che delimitano, in direzione verticale, la sovrapposizione del pixel dell'immagine; $\Delta Y_{pix_{PS}}$ è l'altezza del pixel del PS. Si può trasporre il discorso alla direzione orizzontale calcolando $PSPixXPos_{min}$ e $PSPixXPos_{MAX}$ con le opportune modifiche alle formule.

Adesso, facendo un ciclo in verticale e uno in orizzontale sui soli pixel del PS coperti dal pixel dell'immagine, si calcolano $PSSubPixYPos$ e $PSSubPixXPos$ che sono i valori di copertura parziale, in verticale e in orizzontale, del pixel dell'immagine su quello PS, calcolati come semplici distanze rispetto ai limiti estremi del pixel del PS stesso. Nella figura 7.3, si è cercato di chiarire i concetti ora citati evidenziandone il funzionamento per il pixel identificato con il numero 1 e appartenente ad un PS che si presuppone già individuato tramite i suoi indici.

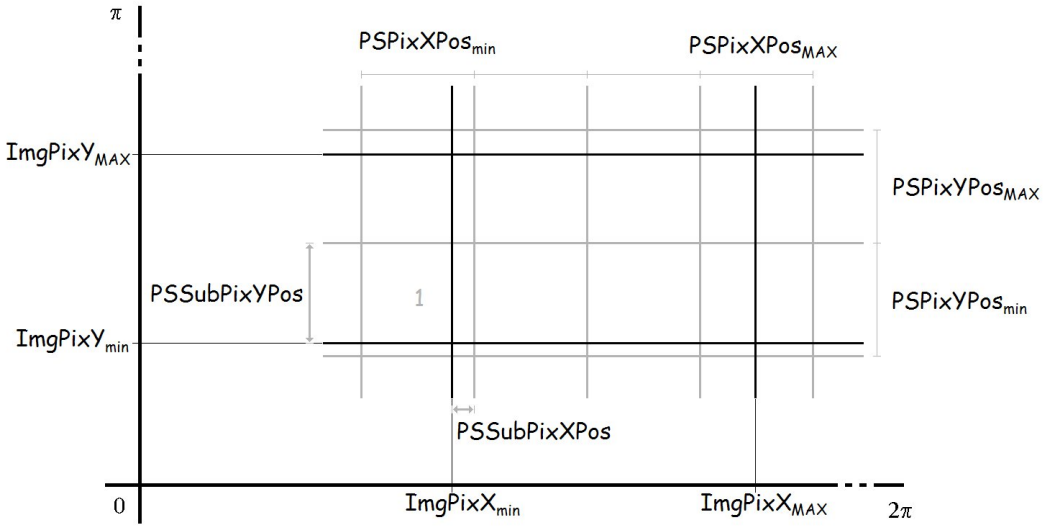

Figura 7.3. Localizzazione e copertura di un pixel dell'immagine NASA (delimitato in nero) rispetto ad uno dei pixel del PS (delimitati in grigio)

A questo punto è possibile chiamare la procedura `MoonPSector::AddPixelInstance()`. Questa, per ogni pixel immagine che rientra nel PS e per il quale deve essere creata la texture, istanzia un elemento di base con le informazioni RGB e la percentuale di copertura del pixel secondo come segue:

$$CoverageRate = \frac{PSSubPixXPos \cdot PSSubPixYPos}{\Delta X_{pix_{PS}} \cdot \Delta Y_{pix_{PS}}}$$

Alla fine dell'elaborazione del PS preso in considerazione, ci si ritrova ad avere una matrice di base con una lista di elementi per ogni pixel. In figura 7.4 viene mostrata la situazione finale per un solo pixel che riceve più contributi e per cui esiste una lista di elementi.

7.10 Impostazione e scrittura della texture del PS

Arrivati a questo punto, in memoria è presente una matrice di dimensioni 256×256 i cui elementi sono strutture [cfr. sezione 7.9] che, in caso di molteplici contributi per un determinato pixel, possono relazionarsi e presentarsi in forma di liste. Grazie alla

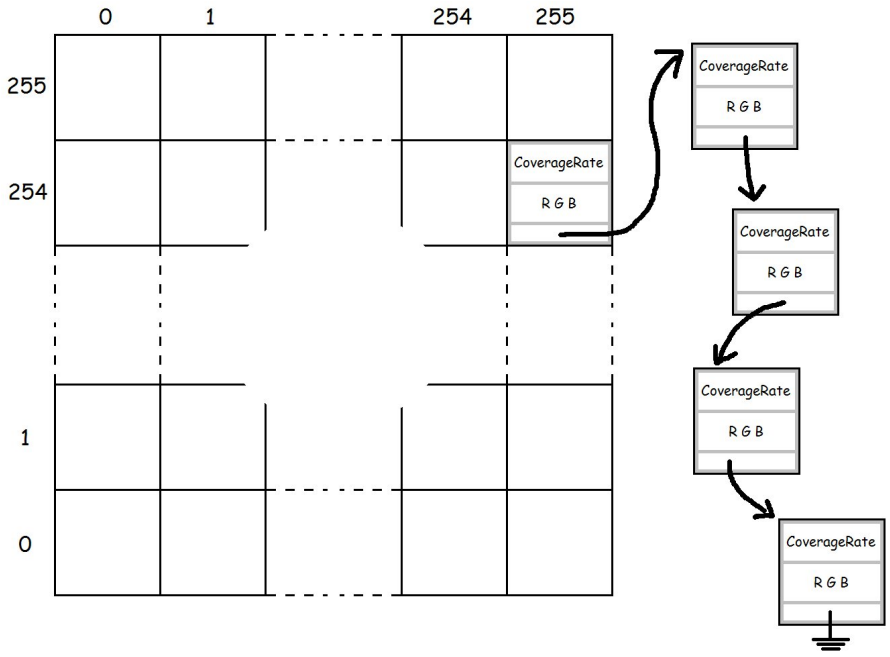

Figura 7.4. Lista di 5 elementi per un generico pixel alla fine della procedura di creazione della texture per un determinato Planet Sector

procedura `MoonPSector::SetPSPixels()` siamo in grado di calcolare i valori RGB di ognuno dei pixel del PS, a partire dalla matrice di elementi di base.
Dopo aver impostato, tramite `MoonPSector::SetOutFullImgName()`, il nome corretto del file grafico da generare, se ne può attivare la procedura di scrittura, e le relative operazioni di settaggio, con `MoonPSector::ManagePSImage()`.
Infine, poichè la matrice degli elementi di base è allocata dinamicamente, è stata prevista la deallocazione della medesima per mezzo di `MoonPSector::DeletePixList()`.

Spiegazione dell'algoritmo

Il punto di partenza è la matrice di strutture di base e si scandiscono tutti gli elementi prima per righe e poi per colonne.
Per ogni pixel, dunque, si controllano tutte le strutture ad esso associato e si fa una media pesata dei contributi secondo la formula

$$\overline{v} = \frac{\sum_{i=1}^{n} p_i \cdot v_i}{\sum_{i=1}^{n} p_i}$$

dove il peso p_i è il valore della copertura superficiale f32CoverageRate dell'i-esima istanza della struttura relativa al pixel in elaborazione, mentre v_i è il valore della componente cromatica. Si faranno, pertanto, i tre calcoli sulle componenti RGB secondo la formula indicata solo per il colore rosso

$$R = \frac{\sum_{i=1}^{n} CoverageRate_i \cdot R_i}{\sum_{i=1}^{n} CoverageRate_i}$$

Quando si raggiunge l'ultimo elemento della lista per quel determinato pixel, il valore della componente è quello definitivo e viene memorizzato nel primo elemento della lista. Alla fine del ciclo su tutti i pixel si ha, così, una matrice di strutture di base in cui il primo elemento è quello valido per settare le componenti del pixel del PS.
Ora con `MoonPSector::SetOutFullImgName()` si imposta il nome corretto del file da scrivere. Esso può risultare di tipo 'raw' se in almeno un elemento della matrice non è associato alcuna struttura e il pixel ad esso associato risulta nero con la codifica (0,0,0); altrimenti sarà di tipo 'bmp' i cui pixel neri avranno la codifica (1,1,1).
Di seguito, con la procedura `MoonPSector::ManagePSImage()` si avviano le operazioni per la generazione del file immagine che rappresenta il PS. Prima di tutto, i valori RGB dei primi elementi della matrice vengono assegnati ad un vettore lineare grande abbastanza per contenere i 256×3 valori RGB dell'intero PS. Infine, con `MoonPSector::WritePSImage()` si genera effettivamente il file, scrivendolo sul disco fisso del computer.

7.11 Accorpamento dei PS

L'accorpamento è l'ultima delle fasi del processo di adattamento delle immagini e prende il via dall'interno di `OrthoPhoto_Package::bProcessCommandLine()` con la procedura `MoonPSector::MergePSsManager()`.
A partire dai PS a massimo dettaglio grafico, si generano i PS padri di livello immediatamente precedente fino ad arrivare all'immagine dell'intero globo di livello 0-A.
All'interno di questo ciclo, settando le opportune variabili e facendo i necessari controlli, è la procedura `MoonPSector::JoinPSs()` ad occuparsi dell'effettivo accorpamento dei settori planetari, coadiuvata necessariamente da `MoonPSector::ResampleGrid()`, che permette di ricampionare più PS in uno singolo, e quando necessario da `MoonPSector::CorrectRaw()`, che consente di correggere in qualche maniera i file 'raw'.

Spiegazione dell'algoritmo

L'algoritmo prevede che si faccia un ciclo su tutti i livelli di dettaglio: il primo ad essere preso in considerazione è quello immediatamente precedente al massimo, per poi decrementare fino ad arrivare allo 0-A.
In seguito si controllano tutti i PS appartenenti al livello preso in considerazione. Per ognuno di loro, si crea una lista degli eventuali sotto-settori, che potrebbero essere accorpati in quello in elaborazione, contandone la presenza, siano essi di tipo 'bmp' o 'raw'.
Se il PS in considerazione esiste già in forma di file 'bmp', si procede alla sua riscrittura solo ed esclusivamente se sono presenti tutti i file 'bmp' dei suoi corrispettivi sotto-settori.
Se, al contrario, il PS in elaborazione non esiste o è rappresentato da un 'raw', si prospettano alcune alternative omologhe:

PS esistente in forma di 'raw' . Se sono presenti solo file di tipo 'bmp' di tutti i suoi sotto-settori, allora viene cancellato il file 'raw' e generato, al suo posto, un file 'bmp'. In alternativa si rigenera il file 'raw' provando a correggere gli eventuali pixel con codifica fallace (0,0,0), avvisando dell'accaduto in un file di resoconto.

PS inesistente . Il nuovo file che rappresenta il PS viene creato se e solo se esiste il numero massimo di file dei sotto-settori, contando congiuntamente quelli buoni e quelli cattivi. Se però esistono solo file 'bmp', verrà creata una texture buona, altrimenti 'raw', non prima di aver tentato di correggere i pixel bacati ed eventulamente riportando l'accaduto nel file di rendiconto.

La generazione dei file avviene per mezzo della procedura `MoonPSector::JoinPSs()`, che si occupa dell'effettiva unione dei settori planetari.

7.12 Unione effettiva dei PS

`MoonPSector::JoinPSs()` unisce realmente i PS interessati nel processo di accorpamento: in un primo momento prevede che i PS interessati vengano accostati formando il PS di interesse ma con le dimensioni ben superiori alle canoniche di 256×256 pixel. Solo in seguito, tale PS temporaneo verrà scalato in uno definitivo dalle dimensioni standard grazie a `MoonPSector::ResampleGrid()`.
L'eventuale entrata in funzione di `MoonPSector::CorrectRaw()` è determinata dal fatto che si stia lavorando con un file cattivo. Tale procedura, infatti, cerca, nei limiti del possibile, di correggere eventuali pixel fallaci (0,0,0) in pixel buoni.
A questo punto il PS è pronto per essere scritto nel file grafico di formato corretto.

Spiegazione dell'algoritmo

Le operazioni avvengono su una matrice con i valori RGB di ogni pixel. Per comodità di accesso diretto, la matrice viene allocata in maniera statica e sovradimensionata per avere la certezza che possa gestire i diversi casi. La massima dimensione verticale è data da due sotto-settori e quindi 2×256 pixel. In orizzontale, invece, il massimo numero di sotto-settori è tre, quindi la matrice, su tale asse, avrà 3×256 pixel.
In base alla posizione sulla superficie, tali valori vengono corrispettivamente collocati nella matrice così da creare un PS unico. Se per esempio si ha in elaborazione un PS di livello 9-J, formato da 4 settori 10-K, allo stato delle operazioni, avremo una matrice dalle dimensioni, in pixel, di $2 \cdot 256 \times 2 \cdot 256$ con tutte le informazioni cromatiche dei pixel dei 4 sotto-settori. Ora con `MoonPSector::ResampleGrid()` si ricampiona l'immagine, ridimensionandola a 256×256, come esemplificato in figura 7.5. Nel nostro piccolo esempio, i pixel del PS provvisorio vengono presi a gruppi di 4 e condensati in uno, calcolando una media pesata delle componenti RGB dei 4 pixel.
Una volta adattata allo standard, l'immagine potrà venir liberata dall'eventuale presenza di pixel indefiniti (0,0,0) richiamando la funzione `MoonPSector::CorrectRaw()`, attivabile per mezzo di uno dei parametri della procedura `MoonPSector::JoinPSs()`. Nel caso si riesca

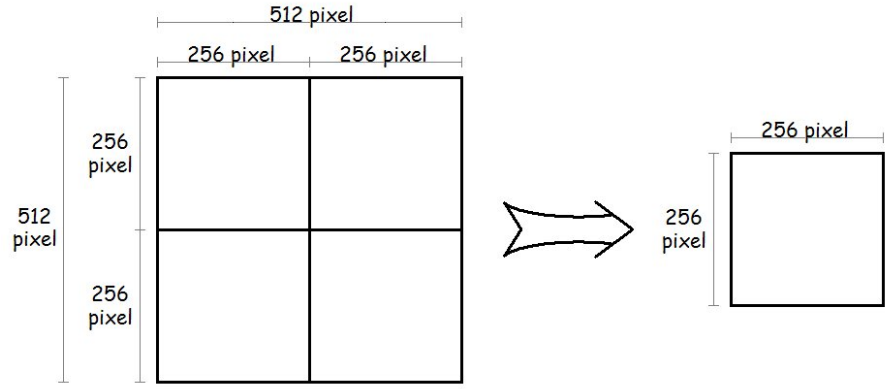

Figura 7.5. Accorpamento dei sotto-settori di livello 10-K in un unico PS di livello 9-J

effettivamente a rendere un file 'raw' in uno 'bmp', si procede anche alla rettifica del nome del file stesso.
Ora, per mezzo di `MoonPSector::SetImageToWrite()` e `MoonPSector::WritePSImage()`, si imposta e si scrive il file che rappresenta il Planet Sector.

Capitolo 8

Classi dell'adattatore

Per questo secondo programma si è pensato di creare un'unica classe C++ che comprendesse tutte le routine necessarie per adattare le immagini del database alle specifiche esigenze del sistema UltraPort per la visualizzazione di ambienti tridimensionali.
Come per l'importatore [cfr. capitolo 5], per non rendere pesante la stesura del lavoro e la sua lettura, ho preferito non riportare il codice nei suoi dettagli implementativi, rinviando la completezza dell'esposizione, e dell'eventuale analisi, direttamente al codice sorgente dell'applicativo medesimo.
Per spiegare le funzioni, però, si è reso necessario almeno la loro proposizione in termini generali, nella speranza di non perdere di vista il filo conduttore dell'intero discorso.
Per il funzionamento, anche l'adattatore necessita delle classi 'ultratools' e 'OrthoPhoto_Package' per le quali si rimanda la visione al capitolo 7.

8.1 La classe MoonPSector

Per comodità espositiva e per continuità proprio con il capitolo 7 riguardante il funzionamento degli algoritmi della procedura, le numerose routine della classe verranno prese in esame seguendo la suddivisione nelle tre parti:

Fase di preparazione che imposta gli elementi di base per la creazione delle cartelle per la memorizzazione dei Planet Sector;

Creazione dei PS di base che si occupa dell'effettiva produzione dei file grafici relativi ai settori planetari di massimo livello di dettaglio;

Accorpamento dei PS che, sostanzialmente, genera i rimanenti Planet Sector per la rappresentazione totale della Luna.

8.2 Fase di preparazione

Questa prima parte della classe 'MoonPSector' è ben rappresentata da cinque routine:

```
void ProcessFile(            const char* pszDirectory);

void UserProcessFile(        const char* pszDirectory,
                             const char* pszFile);

void CreatePSDirs(           const unsigned int u16MaxLevel,
                             const char* pszDirectory,
                             const unsigned int u16ActualLevel,
                             const unsigned int u16XIndex,
                             const unsigned int u16YIndex);

int CalculateMaxPSLvl();

void SetMaxLevelByPixel();
```

ProcessFile()

Questa procedura, utilizzata anche in altre fasi, consente di scandire il contenuto della cartella pszDirectory, alla ricerca di quei file utili al programma.

UserProcessFile()

I parametri pszDirectory e pszFile identificano, rispettivamente, la cartella ed il file, in essa contenuto, da manipolare o da cui acquisire preziose informazioni.
Per come utilizzata in questa fase permette di avviare le operazioni di calcolo dei livelli di dettaglio grafico delle immagini.

CreatePSDirs()

Questa è una procedura recursiva e serve per creare le directory in cui andranno ad essere salvati i file grafici relativi ai PS, scrivendo, inoltre, quelli con le coordinate degli stessi.
La prima operazione svolta è quella di creare la cartella pszDirectory del PS in elaborazione. Questa, denominata con un codice uguale all'identificativo del settore planetario, viene calcolata grazie all'adeguata manipolazione del livello attuale u16ActualLevel in cui si opera e degli indici u16XIndex e u16YIndex che localizzano il PS rispetto alla griglia di settori.
La creazione delle sotto-cartelle prosegue fino a quando non viene raggiunto il valore del livello immediatamente precedente a quello indicato da u16MaxLevel.

CalculateMaxPSLvl()

Fa partire le operazioni per il calcolo dei livelli massimo e minimo di dettaglio, disponibili in relazione al database di immagini dell'adattatore.

SetMaxLevelByPixel()

È la procedura che consente di calcolare effettivamente i livelli di dettagli massimo e minimo.

Le operazioni si basano sul confronto tra le dimensioni dei pixel sia delle immagini, di volta in volta controllate, sia dei Planet Sector calcolati in modo procedurale.

8.3 Creazione dei PS di base

In questa fase si possono riconoscere procedure portanti e altre di supporto, con relativa minore importanza. Tra quelle di base si annoverano le seguenti:

```
void ProcessFile(       const char* pszDirectory);

void UserProcessFile(   const char* pszDirectory,
                        const char* pszFile);

void ManagePSLevel(     const unsigned int u16MaxLevel);

void ManagePSector();

void CreatePSTextures(  const char* pszFileName);

void AddPixelInstance(  const unsigned int u16PSPixX,
                        const unsigned int u16PSPixY,
                        const unsigned int u16NasaPixX,
                        const unsigned int u16NasaPixY,
                        const double f64SubPixX,
                        const double f64SubPixY);

void ManagePSImage();

void SetPSPixels();
```

Le principali routine di supporto, invece, sono:

```
bool bCompareCoord(        const char* pszFile);

bool bCheckDetailLevel();

bool bNasaPixIsInPS(       const unsigned int* pu16CoordX,
                           const unsigned int* pu16CoordY);

void DeletePixList();

void SetOutFullImgName(    const unsigned int u16Level);

int PSFileTypeCounter(     const vector<string> &ACvectorFileList,
                           const unsigned int u16Option);
```

8.3.1 Routine di base

Qui vengono descritte le routine che hanno una fondamentale importanza per lo svolgimento delle operazioni per la creazione dei settori planetari di massimo dettaglio grafico.

ProcessFile()

Come visto nella precedente fase, tale procedura, consente di scandire il contenuto della cartella pszDirectory, alla ricerca di quei file utili al programma.

UserProcessFile()

I parametri pszDirectory e pszFile identificano, rispettivamente, la cartella ed il file, in essa contenuto, da manipolare o da cui acquisire preziose informazioni.
Per come utilizzata in questa fase, permette di avviare sia le operazioni di verifica dell'appartenenza di un PS ad un certo livello di dettaglio, ma anche tutte quelle relative all'adattamento dei PS vero e proprio.

ManagePSLevel()

Gestisce i livelli di dettaglio uno alla volta, a partire da quello identificato da u16MaxLevel fino a quello minimo calcolato con una delle routine di supporto. Viene eseguito un controllo tramite `MoonPSector::bCheckDetailLevel()` per verificare se esiste almeno un file appartenente al livello corrente. In caso di esito positivo, viene avviata la procedura per la manipolazioni dei settori planetari di tutto il livello.

ManagePSector()

Avvia una scansione su tutti i PS appartenenti ad un determinato livello. Inoltre controlla quali file del database si sovrappongono al settore in elaborazione e, in caso di esito positivo, avvia le procedure di acquisizione e creazione del file grafico rappresentante il PS.

CreatePSTextures()

Permette di creare le texture relative ai PS che compongono l'intera superficie lunare. Per mezzo di svariati calcoli e confronti permette di capire dove localizzare l'immagine e come rapportarla ai settori planetari, che poi andranno generati con l'ausilio di procedure come `MoonPSector::AddPixelInstance()`.
La stringa pszFileName è il nome del file da cui vengono acquisiti i valori delle componenti cromatiche da trasferire nei PS.

AddPixelInstance()

Questa è la procedura che associa i valori RGB dei pixel dell'immagine ai corrispettivi del Planet Sector. Va ricordato che per ogni pixel del PS è possibile avere contributi multipli provenienti da più pixel di più immagini del database.
Gli interi u16PSPixX, u16PSPixY, u16NasaPixX e u16NasaPixY, sono gli indici di riga e colonna che identificano sia i pixel dei PS sia quelli delle immagini. I valori f64SubPixX e f64SubPixY, invece, indicano la larghezza e l'altezza, in radianti, dell'area che si sovrappone tra immagine e PS preso in considerazione.

ManagePSImage()

È la procedura che consente di avviare le operazioni di generazione del file rappresentante il PS. Dopo aver copiato i valori RGB del PS in un'apposita struttura in memoria, imposta le proprietà fisiche del file grafico, ne setta i valori cromatici per ogni pixel e infine lo crea effettivamente su disco.

SetPSPixels()

Copia i valori cromatici, dei pixel del PS, da un'aria di memoria ad un'altra di più immediato utilizzo da parte delle procedure per la generazione dei file grafici.

8.3.2 Routine di supporto

Procedure e funzioni riportate qui di seguito, ricoprono un ruolo secondario, ma comunque importante.

bCompareCoord()

Questa funzione booleana consente di confrontare le coordinate del file pszFile con quelle di un PS. Se l'immagine si sovrappone anche solo parzialmente al settore planetario, viene restituito il valore TRUE; FALSE altrimenti.

bCheckDetailLevel()

Funzione utilizzata per indicare se una data immagine può essere presa in considerazione per la creazione dei PS ad un determinato livello di dettaglio: restituisce TRUE in caso positivo, FALSE altrimenti.

bNasaPixIsInPS()

Permette di capire se un determinato pixel di un'immagine del database rientra, anche solo parzialmente, nel PS in fase di elaborazione.

DeletePixList()

Questa è la procedura che si occupa di liberare la memoria, e renderla quindi nuovamente a disposizione, deallocando la matrice di elementi di base in cui vengono salvate le informazioni cromatiche per la generazione dei settori planetari.

SetOutFullImgName()

Imposta in modo corretto il nome del file che dovrà essere generato.

PSFileTypeCounter()

Funzione che, per un dato PS, consente di calcolare il numero dei suoi sotto-settori rappresentati da file 'bmp' o 'raw'.
L'intero u16Option indica il tipo di file, mentre ACvectorFileList è la lista dei possibili sotto-settori esistenti e che bisogna contare.

8.4 Accorpamento dei PS

Anche per questa fase si possono individuare delle procedure fondamentali e altre che aiutano nello svolgimento delle operazioni di accorpamento.
Quelle basilari sono tre:

```
void MergePSsManager();

void JoinPSs(            const vector<string> &ACvectorFileList,
                         const bool bRawManipulation);

void ResampleGrid(       unsigned char mu8Grid[][SECTOR_SIDE_PIXEL*9]);
```

Quelle secondarie invece sono:

```
void SetImageToWrite(     const enum eImageFileType eType);

void SetImageToWrite(     const enum eImageFileType eType,
                          const unsigned int u16Width,
                          const unsigned int u16Height);

void WritePSImage();

void SetOutFullImgName(   const unsigned int u16Level);

void PSFileTypeCounter(   const vector<string> &ACvectorFileList,
                          const unsigned int u16Option);

void GetIndexByFileName(  const char* pszFileName,
                          unsigned int &u16IndexX,
                          unsigned int &u16IndexY);

void CorrectRaw(          const char* pszFileName,
                          unsigned char mu8Grid[][SECTOR_SIDE_PIXEL*9]);
```

8.4.1 Procedure di base

Vengono qui presentate quelle procedure che assumono un ruolo primario nell'accorpamento, a partire dai PS di livello massimo di dettaglio.

MergePSsManager()

Sostanzialmente, tale procedura ricopre il ruolo di gestore delle operazioni di accorpamento dei PS esistenti, per mezzo di un ciclo che parte dal livello di dettaglio massimo fino al

livello minimo 0-A.
La prima operazione svolta è quella di settare le variabili di grande importanza come, ad esempio, nomi di file e cartelle con relativi percorsi. A questo punto, dopo aver fatto opportuni controlli, può far avviare la procedura che permette di aggregare più PS in uno unico con minore livello di dettaglio.

JoinPSs()

Questa è la procedura che si occupa dell'effettivo accorpamento di più settori planetari in uno unico e l'intera fase avviene essenzialmente in due parti.
La prima consiste nel creare, in memoria, un aggregato dei PS di interesse semplicemente affiancandoli l'un l'altro, senza tenere conto delle esigenze di UltraPort. Solo secondariamente, si provvede a ridimensionare il grande "collage" così ottenuto, al fine di avere un file 'bmp' di 256 pixel per lato.
La lista ACvectorFileList contiene i nomi dei file dei corrispettivi PS che dovranno essere accorpati, mentre la variabile boolena bRawManipulation indica, se impostata a TRUE, che è necessario provare a correggere un file 'raw' nel tentativo di renderlo buono, e quindi 'bmp', o comunque di migliorarlo in qualche parte.

ResampleGrid()

Il suo ruolo è quello di ridimensionare la griglia mu8Grid in argomento. Questa è una matrice che contiene i valori RGB di tutti i pixel dei settori planetari che sono stati affiancati, creando un'immagine in memoria più grande dei 256 pixel per lato richiesti da UltraPort.
Alla fine delle operazioni, si avrà una griglia utile di 256 pixel sia in orizzontale che in verticale, contenente tutte le informazioni necessarie per la creazione del file grafico che rappresenta il PS da generare e da scrivere su disco.

8.4.2 Procedure di supporto

Le procedure che seguono, ricoprono un ruolo minore rispetto alle precedenti, ma pur sempre necessario allo scopo da raggiungere.

SetImageToWrite()

Come si vede, la procedura, che si occupa di impostare il tipo e le dimensioni del file grafico da generare, è presente in due versioni. In una, oltre al tipo del file identificato tramite l'argomento enumerativo eType, vengono esplicitamente dichiarate le dimensioni, in pixel, orizzontale u16Width e verticale u16Height, che altrimenti vengono impostate per default a 256.

WritePSImage()

Si occupa della scrittura su disco del file grafico che rappresenta il Planet Sector in elaborazione.

SetOutFullImgName()

Imposta in modo corretto il nome del file che dovrà essere generato.

PSFileTypeCounter()

Funzione che, per un dato PS, consente di calcolare il numero dei suoi sotto-settori rappresentati da file 'bmp' o 'raw'.
L'intero u16Option indica il tipo di file, mentre ACvectorFileList è la lista dei possibili sotto-settori esistenti e che bisogna contare.

GetIndexByFileName()

Consente di carpire, dal nome del file del PS pszFileName, gli indici numerici u16IndexX e u16IndexY che ne identificano la posizione sull'intero scacchiere di settori planetari in relazione al livello corrispondente.

CorrectRaw()

Questa procedura ha il compito di correggere i pixel non buoni del file 'raw' pszFileName fino a trasformarlo, compatibilmente con i dati a disposizione, in un file 'bmp' buono, operando sulla griglia mu8Grid che rappresenta l'immagine del PS. Il suo ruolo è quello di sostituire i valori delle componenti cromatiche dei pixel indefiniti, dunque con codifica (0,0,0), con altri validi, se possibile.
Nel caso si riesca a eliminare tutti i pixel indefiniti, si procede al cambiamento del nome del file che, ad esempio, da 'M_3_2_raw.bmp' passerà al più semplice e corretto 'M_3_2.bmp'.

Conclusioni

L'esperienza tesistica svolta presso la Fondazione Ultramundum assume un significato molto importante per quanto concerne la crescita formativa in ambito informatico.
Normalmente, durante il percorso universitario, si ha una conoscenza di base delle tecniche di programmazione, alle volte decontestualizzato rispetto alle realtà di ricerca e sviluppo software. Il relazionarsi con problematiche ad ampia visuale per la realizzazione di progetti ambiziosi come quelli proposti dalla Fondazione, ha reso possibile la personale crescita culturale non solo in ambito prettamente connesso alla programmazione, ma anche in quello relativo alla grafica tridimensionale.
Il ruolo del PID è fondamentale proprio per avere permesso da una parte di approfondire le conoscenze di programmazione in ambito C/C++, dall'altra di acquisire quelle nozioni di base sulla grafica digitale 2D e 3D che ampliano il personale pacchetto culturale ponendo le basi per un eventuale approfondimento con un minimo di cognizione di causa.

Confronto tra gli scopi prefissi ed i risultati ottenuti

Il PID della Luna si è rivelato un progetto decisamente ambizioso e di non facile creazione. Il fatto di essere riusciti nell'intento rispettando i propositi iniziali, porta grande soddisfazione e a livello personale e aziendale.
Si può dire che il plastico interattivo digitale creato potrebbe rappresentare un viatico importante, ma sicuramente non decisivo, per far accrescere nell'opinione comune le possibilità che la Fondazione offre in campo informatico e tecnologico.

Commento critico dei risultati ottenuti

Il prodotto grafico interattivo, così come esposto in valutazione di tesi e disponibile in futuro sul sito della Fondazione medesima, è il risultato di parecchi sforzi sia in ambito di ricerca che di implementazione.
Il reperimento di dati scientifici di una certa valenza non è stato così complicato quanto la loro interpretazione e il loro utilizzo. Nonostante la massima reputazione della NASA in campo aerospaziale, una gran parte dei dati a disposizione si è rivelata un po' deficitaria riguardo la qualità di molte delle immagini.
Il PID lunare si pone in competizione con i più diffusi applicativi realizzati quali *Celestia* e *Google Moon*, ma soprattutto con *World Wind* realizzato dalla NASA stessa: l'essere

arrivati a "competere" ai livelli dell'agenzia americana regala quel senso di soddisfazione che si prova nel portare a termine compiti di assoluta importanza anche al di fuori dei confini nazionali.
L'importanza della realizzazione del PID proposto con questa tesi, è bene rappresentato dal confronto con le immagini di World Wind per le stesse aree.

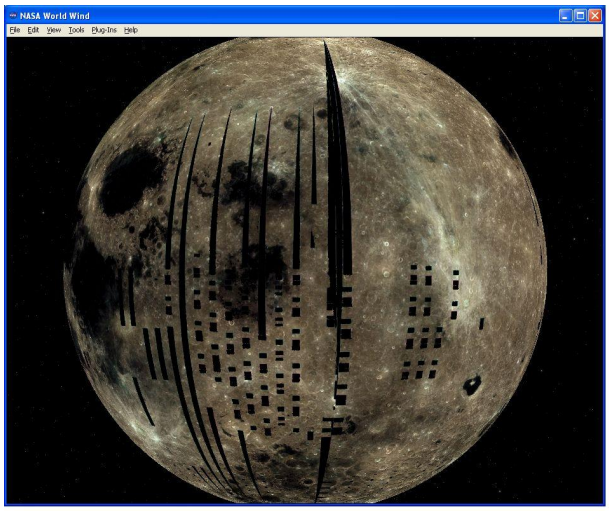

Figura 8.1. Esempio 1-A. Visualizzazione panoramica della Luna con World Wind della NASA

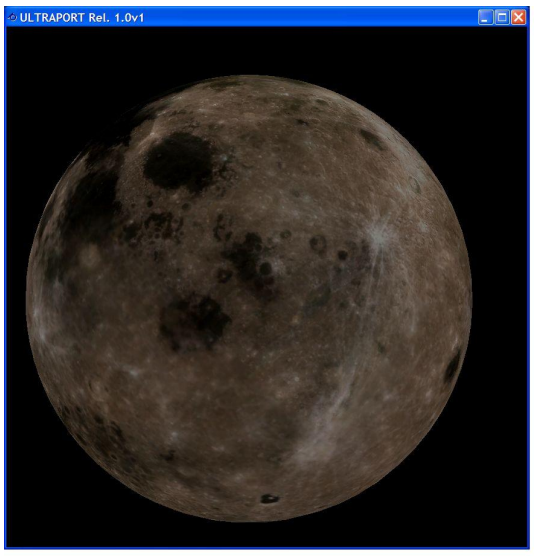

Figura 8.2. Esempio 1-B. Visualizzazione panoramica della Luna con Ultra-Port di Ultramundum

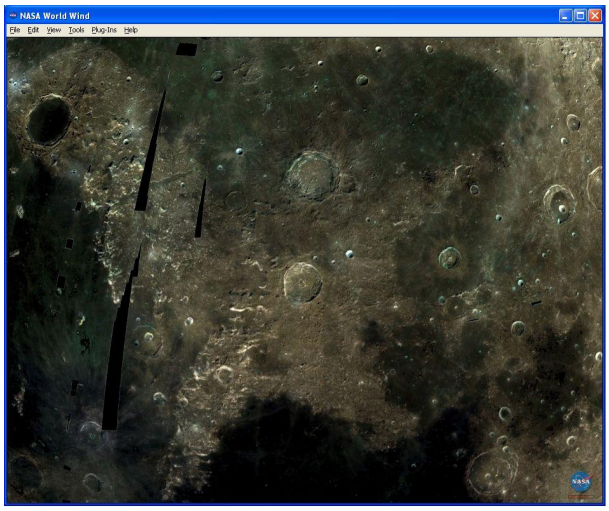

Figura 8.3. Esempio 2-A. Visualizzazione di una particolare area con World Wind

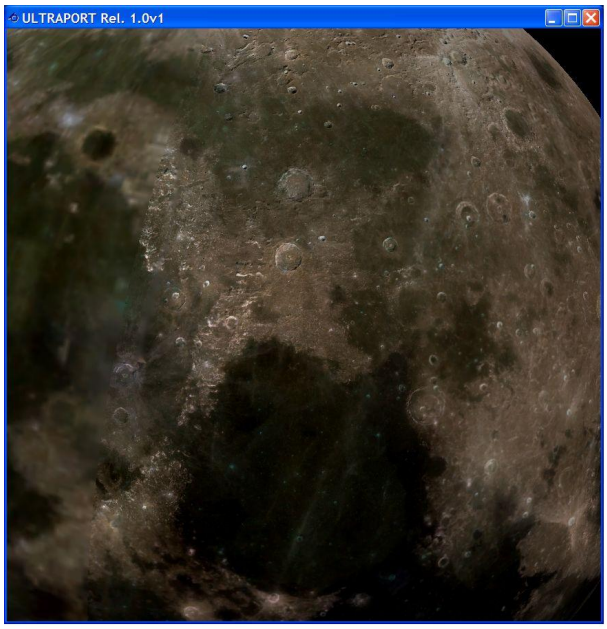

Figura 8.4. Esempio 2-B. Visualizzazione di una particolare area con UltraPort

Possibili ulteriori sviluppi della ricerca

Il PID è un bel progetto che ha richiesto molto tempo e preparazione. Lo sforzo profuso, però, non è stato focalizzato alla sola realizzazione del medesimo, bensì al creare anche una serie di classi che permettano di avere un punto di partenza per la creazione di futuri plastici digitali interattivi.
Non si esclude che il codice sorgente scritto presenti imperfezioni e sicuramente può essere migliorato, però può essere il punto di partenza per eventuali sviluppi futuri di probabili altri importatori grafici di dati così particolari come quelli messi a disposizione dalla NASA.

Appendice A

Elenco dei simboli

Viene qui presentata una lista riepilogativa degli acronimi utilizzati più di frequente in tutto il lavoro svolto.

AMMOS - Advanced MultiMission Operations System;

ASCII - American Standard Code for Information Interchange;

BMDO - Ballistic Missile Defense Organization;

CCD - Charge Couple Device;

CIE - Commission Internationale dell'Enclairage;

DIM - Digital Image Model;

DOD - Department Of Defense;

DSPSE - Deep Space Program Science Experiment;

EDR - Engineering Data Record;

ESA - European Space Agency;

HIRES - HIgh RESolution (camera);

ISIS - Integrated Software for Imagers and Spectrometers;

L3 - Label Library Lite;

LIDAR - Laser Image Detection And Ranging (system);

LWIR - Long Wavelength InfraRed (camera);

MGDS - Multimission Ground Data System;

MSB - Most Significant Byte;

NASA - National Aeronautics and Space Administration;

NIR - Near InfraRed (camera);

OAL - Object Access Library;

ODL - Object Description Language;

PDS - Planetary Data System;

PID - Plastico Interattivo Digitale;

PS - Planet Sector;

PSDD - Planetary Science Data Dictionary;

SEA - Sinusoidal Equal-Area;

SFOC - Space Flight Operations Center;

USGS - United States Geological Survey;

UVVIS - UltraViolet/VISible (camera);

Bibliografia

[1] Fondazione Ultramundum, *http://www.ultramundum.org*

[2] NASA, *http://www.nasa.gov*

[3] ESA, *http://www.esa.int*

[4] Celestia: the free space simulation, *http://www.shatters.net/celestia*

[5] NASA World Wind, *http://worldwind.arc.nasa.gov*

[6] Google Earth, *http://earth.google.com*

[7] Google Moon, *http://www.google.com/moon*

[8] PDS Imaging Node, NASA, *http://pds-imaging.jpl.nasa.gov/Missions/Clementine_mission.html*

[9] *Planetary Data System Standards Reference*, JPL D-7669 Part 2 Version 3.7, Jet Propulsion Laboratory, California Institute of Technology, Pasadena (California, USA), Marzo 2006, *http://pds.jpl.nasa.gov/documents/sr/StdRef_20060320.v3.7.pdf*

[10] AA.VV., *Planetary Science Data Dictionary Document*, JPL D-7116 Rev. E, Jet Propulsion Laboratory, California Institute of Technology, Pasadena (California, USA), Agosto 2002, *http://pds.jpl.nasa.gov/documents/psdd/psdd.pdf*

[11] R.Davis, S.Monk, *Object Access Library User's Guide*, OAL version 1.8, Laboratory for Atmospheric and Space Physics University of Colorado, Boulder (Colorado, USA), Dicembre 1997, distribuito assieme al software

[12] M.Cayanan, *PDS Label Library Light (L3) User's Guide*, draft version 1.0, Jet Propulsion Laboratory, California Institute of Technology, Pasedena (California, USA), Gennaio 2006, distribuito assieme al software

[13] USGS - ISIS, *http://isis.astrogeology.usgs.gov/Isis2/isis-bin/isis.cgi*