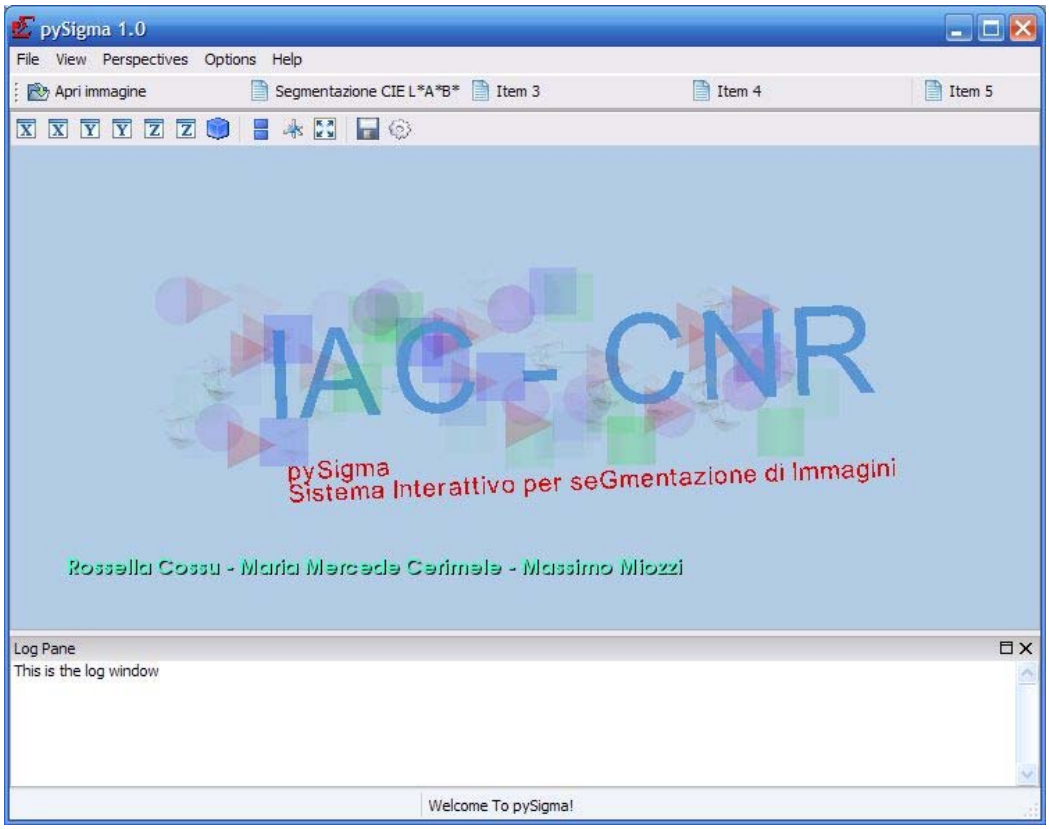
pySIGMA

Interfaccia python per SIGMA

Sistema Interattivo per seGmentazione di imMAgini



Settembre 2008

Massimo Miozzi

Contratto N. 0000729 del 16/06/2008

Istituto per le Applicazioni del Calcolo – IAC – CNR

Indice

1 Introduzione 3
2 Strumenti utilizzati 4
3 La libreria pySigma (C++) 5
3.1 Costruttori, distruttori e il metodo GetOutput 8
3.2 L'allocazione della memoria e il metodo SetImage 10
3.3 Le interfaccia pySigma.i, numpy.i e il passaggio di dati tra Python e C++ 10
4 L'ambiente Python 13
4.1 Il gestore della grafica 13
4.2 Il gestore delle immagini 16
4.3 Il gestore delle interattività 17
4.4 Il gestore delle operazioni 18
5 Inserimento di nuove funzionalità in pySigma 19
5.1 Inserimento di una classe c++ 19
5.2 Inserimento di una attività 19
6 Manuale d'uso 20

1 Introduzione

Questo rapporto costituisce la relazione finale di cui all'art. 4 del contratto di prestazione d'opera in regime di lavoro autonomo occasionale tra l'Istituto per le Applicazioni del Calcolo "Mauro Picone" e l'Ing. Massimo Miozzi (N. 0000729 del 16/06/2008).

In questo documento vengono descritte le modalità di riorganizzazione della libreria relativa al sistema di segmentazione Sigma (Sistema Interattivo per la seGmentazione di imMagini), il processo per la generazione di moduli (wrapper) Python a partire dai codici sviluppati in ambiente C++ e infine l'ambiente Python che gestisce l'interfaccia grafico e la successione delle operazioni di calcolo. Vengono inoltre riportate alcune semplici linee guida per l'inserimento di nuove routines, dalla scrittura della classe C++ al suo utilizzo mediante un nuovo pannello di comando. Un manuale d'uso allo stato attuale del codice completa il rapporto.

Lo scopo del lavoro consiste nella generazione di una libreria a collegamento dinamico (_pySigma.pyd), contenente tutte le funzionalità implementate nelle classi Sigma (C++), utilizzabile da un ambiente Python attraverso un modulo di wrapper (pySigma.py). Tale ambiente è costituito da un gestore per l'esposizione di comandi e finestre, da un gestore di immagini per l'apertura di files e la eventuale manipolazione di base, da un gestore di scena che fornisce l'interattività dell'interfaccia grafico e da un gestore del calcolo per eseguire l'analisi.

L'utilizzo di un contesto di scripting non pregiudica l'efficienza del calcolo, in quanto la totalità delle routine numeriche è contenuta nelle librerie C++, mentre all'ambiente Python viene delegata unicamente la selezione delle operazioni e l'assegnazione dei parametri. In altre parole, esso svolge il ruolo di collante tra differenti contesti.

Nonostante la complessità dello schema proposto possa sembrare elevata, l'approccio modulare con cui è stato organizzato il codice consente di effettuare modifiche e/o aggiunte in modo molto semplice, concentrando le attività unicamente sul segmento di interesse. Si raccomanda di effettuare modifiche sul codice solo dopo aver considerato attentamente l'organizzazione dello stesso.

2 Strumenti utilizzati

Vengono ora elencati gli strumenti utilizzati per la realizzazione del sistema pySigma.

Per la compilazione del codice C++ sono stati predisposti due ambienti di lavoro, pySigma.dsw e pySigma.sln, contenuti rispettivamente nelle cartelle <$HOME> <MSVC6> e <$HOME> <VS2005>. Nonostante la compilazione possa essere effettuata indifferentemente mediante i due strumenti, si consiglia di utilizzare l'ambiente VS2005, in quanto quest'ultimo garantisce una maggior efficienza del codice compilato. E' possibile anche ricompilare le librerie in ambiente Linux, ma tale possibilità al momento non è supportata. I files sorgente, contenuti nelle cartelle <$HOME> <pySigma> <src> (file *.cpp), <$HOME> <pySigma> <include> (*.h) e <HOME><pySigma><swig> (*.i) sono gli stessi per qualunque compilatore.

Per l'ambiente Python si è scelto di utilizzare la distribuzione EPD di Enthought, ver. 2.5.2001 (http://www.enthought.com/). Tale distribuzione, oltre a garantire le consuete caratteristiche dell'ambiente Python (open source, multipiattaforma etc), mette a disposizione in modo integrato un numero significativo di librerie (oltre 60), alcune delle quali sono state utilizzate in pySigma. Più precisamente:

wxPython, toolkit multipiattaforma per la gestione di GUI in ambente Python. http://wxpython.org/download.php.

VTK, Visualizzation Toolkit

TVTK, Traited VTK, wrapper in python per VTK

CHACO2, applicazione per il plot scientifico in python

Numpy, libreria per il calcolo scientifico.

Altri strumenti utili per una comprensione di base e per l'editing grafico e di testo sono:

wxPython demos and docs. Eccellente raccolta di demo per la gestione di controlli in wxPython.

Boa Constructor, GUI developer molto efficiente nella gestione delle finestre grafiche. http://boa-constructor.sourceforge.net/

SPE IDE - Stani's Python Editor. Editor per Python con un efficiente autocompletamento di proprietà e metodi. http://pythonide.blogspot.com/

3 La libreria pySigma (C++)

Lo schema di flusso per la generazione del modulo pySigma è riportato in Figura 3-1. Le configurazioni disponibili all'interno dei workspace sono quelle di Debug e di Release. Le librerie possono essere utilizzate dall'eseguibile generato dal progetto Main, contenuto anch'esso nel workspace. La configurazione di Release genera il modulo pySigma, costituito dalla libreria _pySigma.pyd e dal file di wrapper pySigma.py.

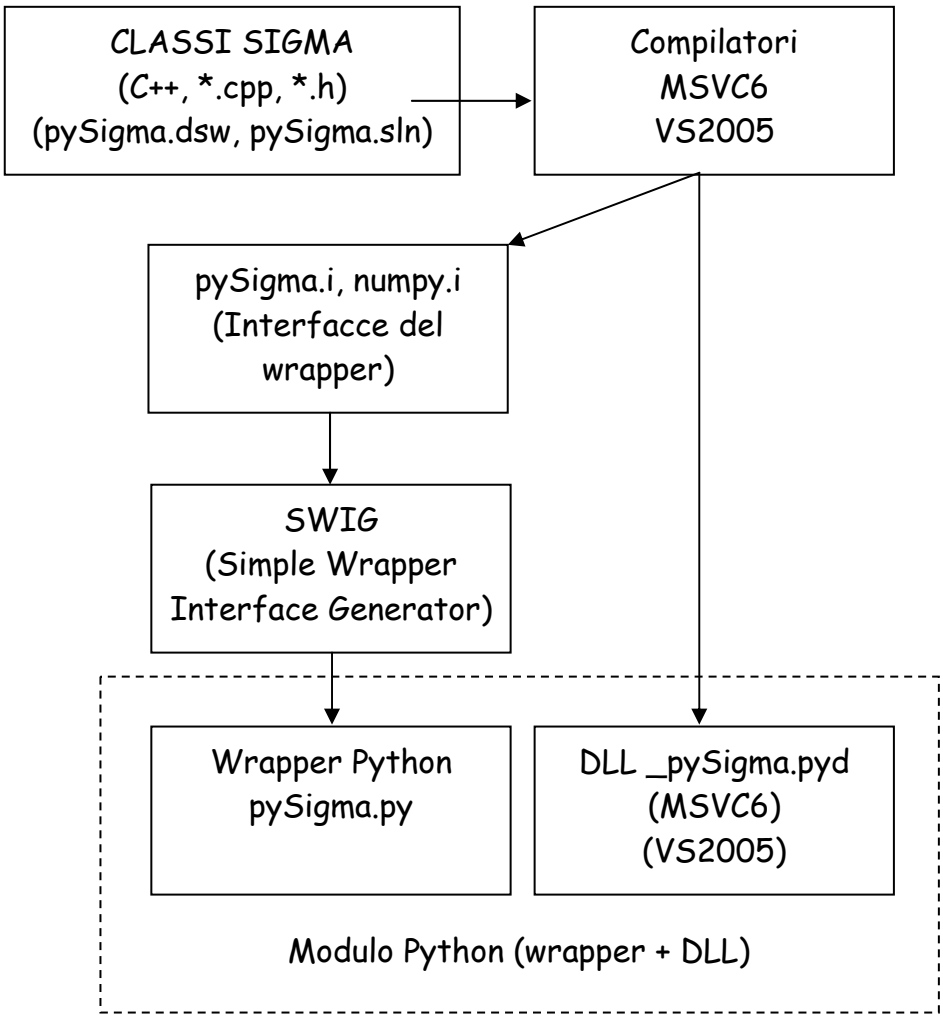


Figura 3-1 Schema di flusso per la generazione della Dynamic Linked Library _pySigma.pyd e del modulo di wrapping pySigma.py a partire dalla libreria Sigma (C++) e dal file di definizione delle interfacce del wrapper pySigma.i

La necessità di garantire una corretta gestione dell'utilizzo della memoria all'interno della libreria Sigma e nei suoi rapporti con l'ambiente Python ha imposto una revisione del codice preesistente. La libreria Sigma è stata riorganizzata definendo tre tipi di classi base e generando le altre classi come derivate da una di queste. Le tre classi base sono:

- Image.cpp
- LABImage.cpp

- RGBImage.cpp

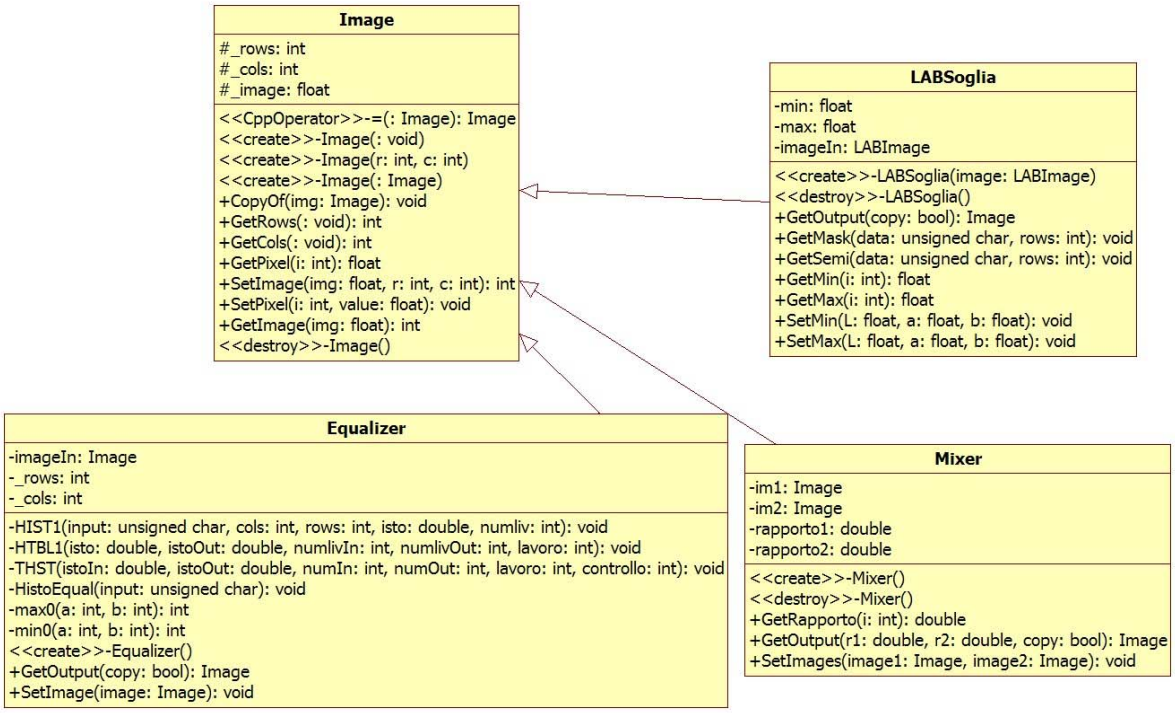


Figura 3-2: LABSoglia, Mixer ed Equalizer derivate da Image

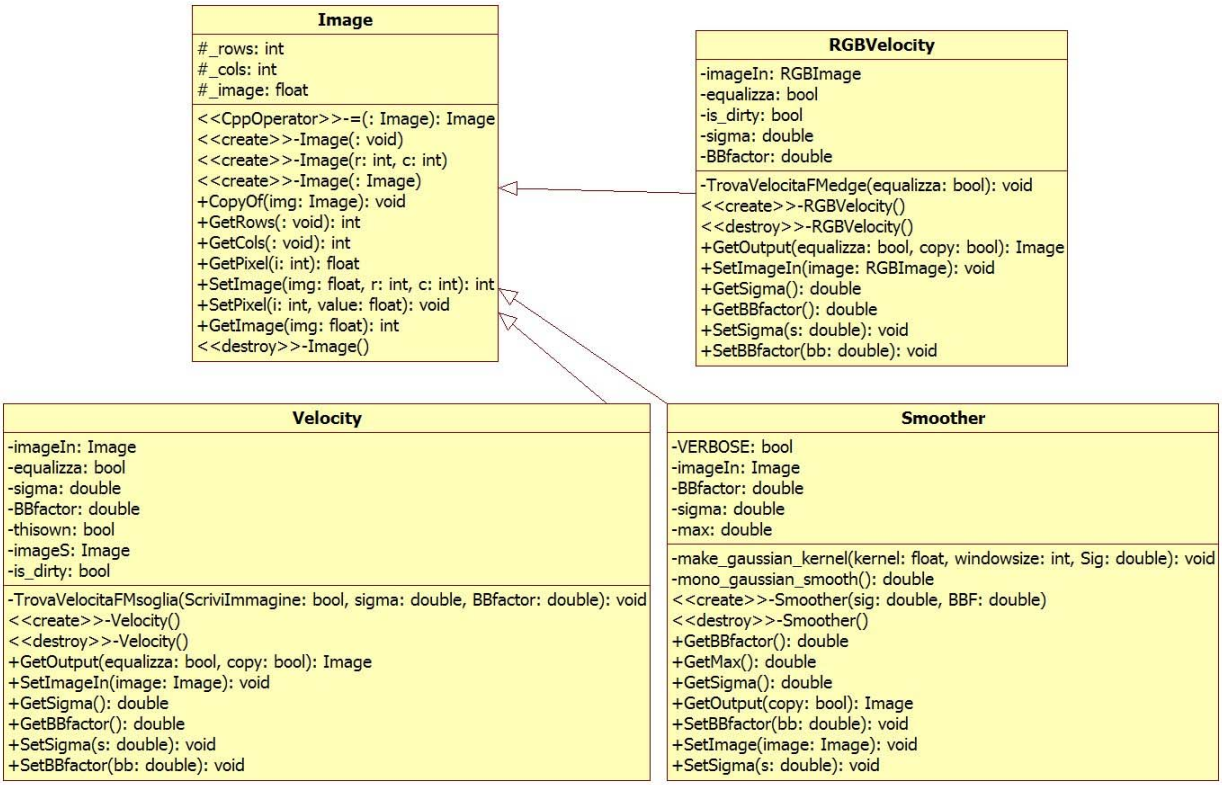


Figura 3-3: RGBVelocity, Velocity, Smoother derivate da Image

Tutte le altre classi sono derivate da una di queste e ne ereditano le proprietà e i metodi. Il tipo di classe base è stabilito in base al tipo di classe restituito dal metodo

GetOutput, comune a tutte le classi derivate. Le classi derivate da Image sono riportate in Figura 3-2, Figura 3-3 e Figura 3-4.

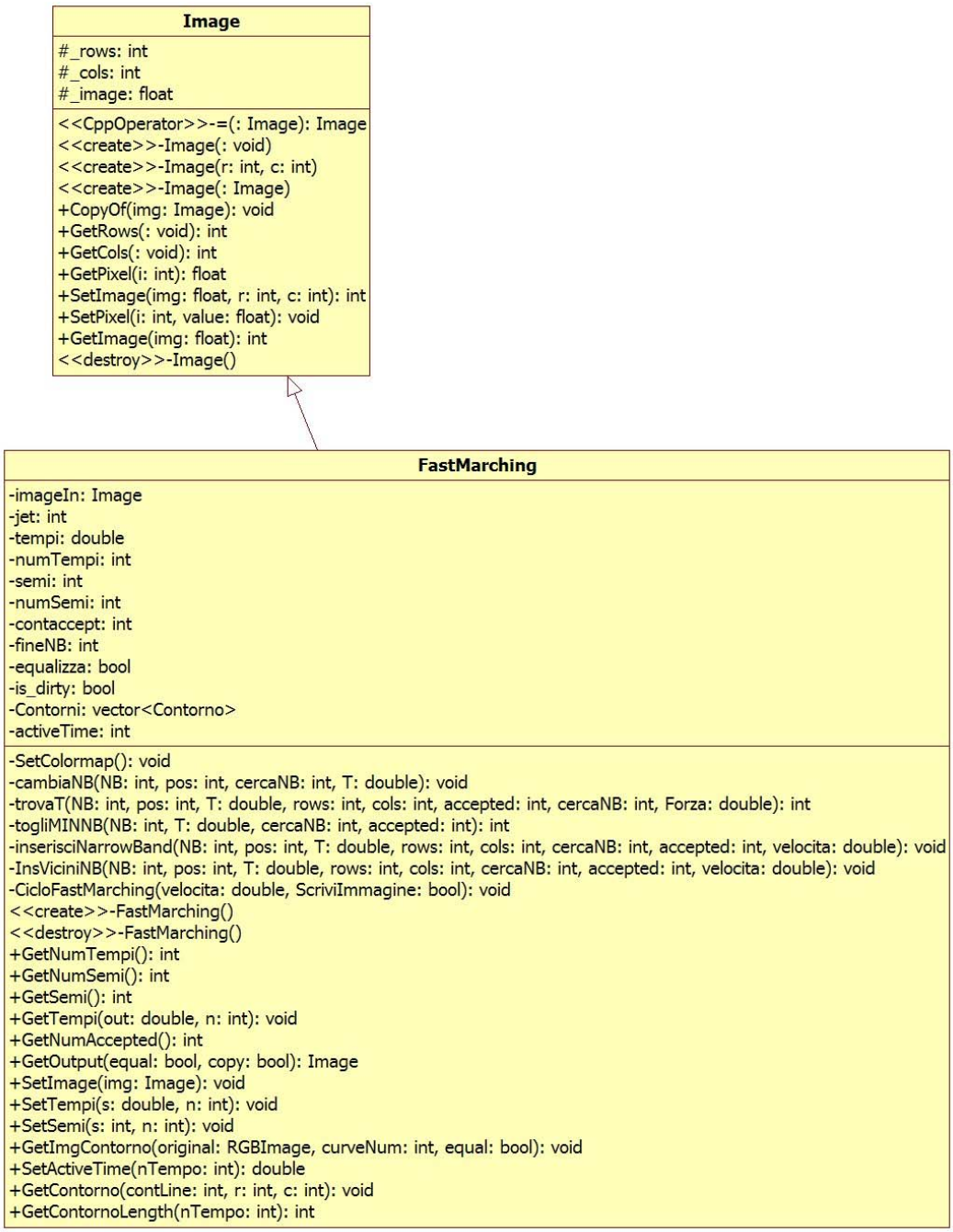


Figura 3-4 FastMarching derivata da Image

La classe derivata da LABImage è riportata in Figura 3-5. Le classi derivate da RGBImage sono riportate in Figura 3-6.

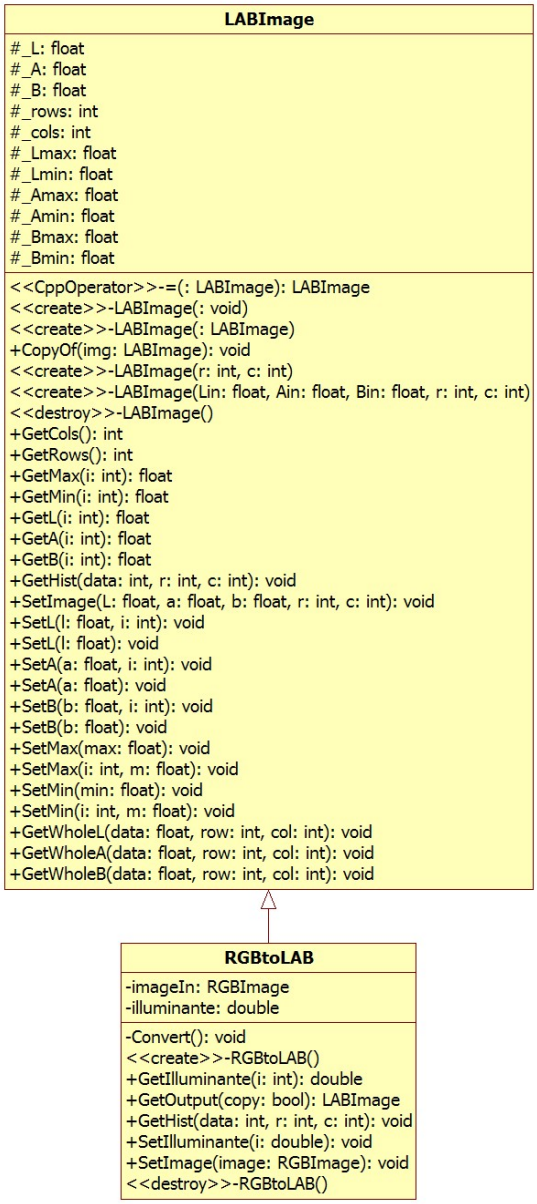


Figura 3-5: RGBtoLAB derivata da LABImage

3.1 Costruttori, distruttori e il metodo GetOutput

Oltre ai metodi costruttori già presenti, tutte le classi base sono state dotate di un costruttore di copia e di un assegnatore di copia. Il primo è un metodo pubblico, il secondo è dichiarato come metodo privato ed esposto dall'interfaccia CopyOf. Questo approccio permette una corretta generazione del wrapper mediante SWIG.

L'applicazione di questi metodi alla classe Image è riportata in Figura 3-7.

I metodi di costruzione ed assegnazione di copia sono stati introdotti per eliminare le ambiguità presenti nel metodo GetOutput, comune a tutte le classi derivate, il quale invece di restituire all'esterno un puntatore a una classe definita come proprietà di una classe contenitore, restituisce ora una copia della classe base da cui la classe

contenitore eredita proprietà e metodi. Un eventuale rilascio della memoria della classe contenitore o di quella appartenente alla copia della classe base non comporta più alcun conflitto e può essere effettuata indifferentemente sia nell'ambiente C++ che in quello Python.

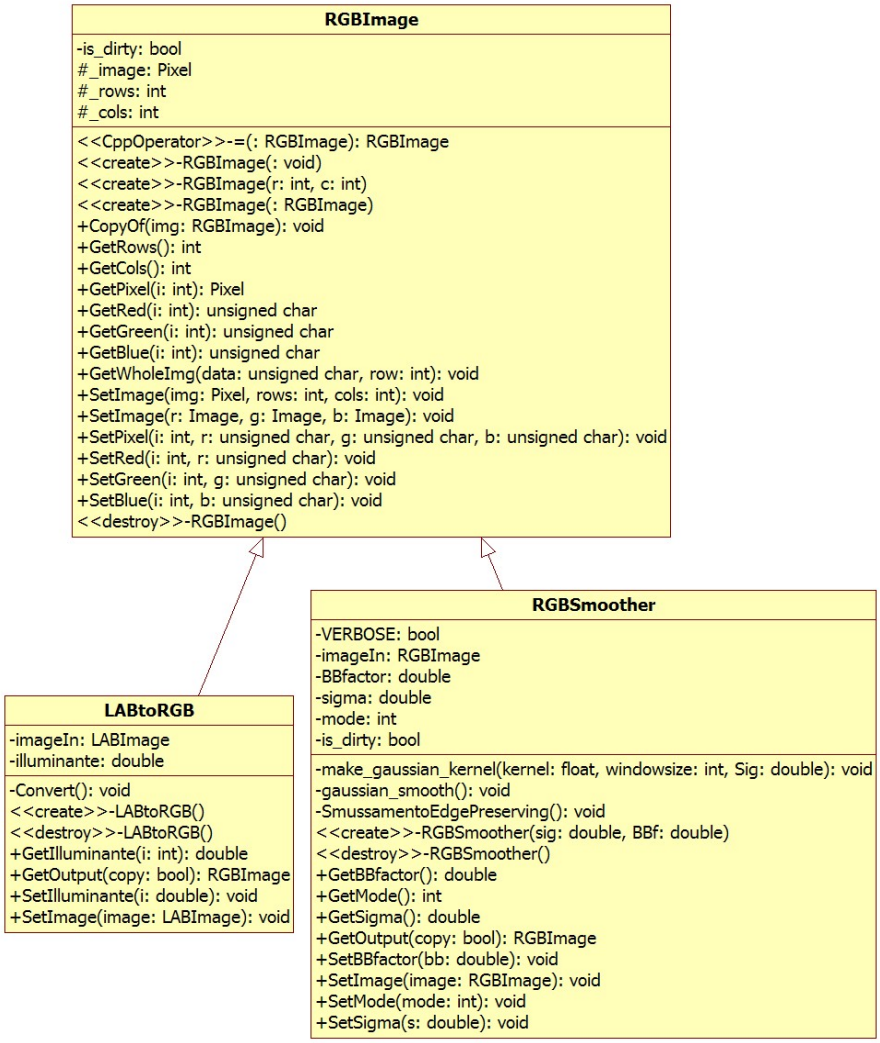


Figura 3-6 LABtoRGB e RGBSmoother derivati da RGBImage

```
    private:
          Image& operator=(const Image&);     //copy assignator
    public:
          Image(const Image&);    //copy constructor
          void CopyOf(Image* img);            //public copy assignator
```

Figura 3-7 Costruttore di copia ed assegnatore di copia per la classe Image

Ogni classe è stata dotata di un metodo distruttore che si occupa di rilasciare la memoria precedentemente allocata all'interno della classe. Nelle classi derivate, il metodo distruttore è dichiarato *virtual*, in modo da premettere l'esecuzione del metodo distruttore della classe base, cui la classe derivata fa riferimento.

3.2 L'allocazione della memoria e il metodo SetImage

Nella riscrittura della libreria Sigma, sono state rimosse in modo quasi definitivo le chiamate di allocazione a malloc e calloc. Queste sono state sostituite dalle più efficaci new, new [], delete e delete[].

Le classi derivate effettuano l'elaborazione dei dati assegnati attraverso il metodo SetImage. La memoria relativa alla classe base viene assegnata all'interno del metodo SetInput. Questa regola pratica permette di individuare rapidamente il luogo della allocazione dinamica, evitando riassegnazioni o rilasci indesiderati.

3.3 Le interfaccia pySigma.i, numpy.i e il passaggio di dati tra Python e C++

L'utilizzo di SWIG, Simple Wrapper Interface Generator, consente di generare un file di wrapper il quale, importato in ambiente Python, può accedere ai contenuti della DLL cui fa riferimento.

Tale meccanismo si basa sulla inclusione, nel progetto della libreria, di un file di interfaccia pySigma.i e di un file (inizialmente vuoto) pySigma_wrap.cxx. Al progetto va aggiunto un comando di post-built, che nel caso del workspace in VS2005 è il seguente:

```
echo on
swig -c++ -python -outdir ../release/vs2005 $(InputPath)
```

Ovviamente l'eseguibile swig.exe deve essere reso visibile all'interno del compilatore, aggiungendo la cartella che lo contiene al path dello stesso.

Durante la compilazione il codice genera il contenuto del file pySigma_wrap.cxx e infine il modulo di wrap pySigma.py.

All'interno dell'ambiente Python, la classe pySigma è richiamabile mediante il meccanismo di import:

```
import pySigma
```

a seguito del quale tutte le classi della libreria sono immediatamente disponibili. Ad esempio, per la classe Image, si potrà avere la successione di comandi riportata in Figura 3-8, nella quale viene inizialmente resa visibile in Python la cartella dove risiede il modulo, questo viene poi importato e ne vengono utilizzate le proprietà e i metodi.

In particolare nel frammento di script, l'oggetto *img* è una istanza della classe ps.Image generata dal costruttore che assegna righe e colonne, *img1* è l'oggetto ottenuto come istanza della classe ps.Image mediante il costruttore di copia su *img* e *img2*, istanziata come vuota, viene riempita assegnandole una copia di *img1*. La memoria dei tre oggetti è separata e ben definita e viene automaticamente rilasciata

secondo il meccanismo della garbage collection di Python oppure assegnando all'oggetto il valore None.

```
>>> import sys
>>> sys.path.append('D:\\lavoro\\iac\\pySigma\\Release\\vs2005')      #Rende visibile la cartella contenente
                                                                     # _pySigma.pyd e pySigma.py
>>> import pySigma as ps                                           #Importa il modulo pySigma
>>> img = ps.Image(1000,1000)                                      #Costruttore di immagine 1000x1000
>>> img.GetRows()                                                  #Chiamata a metodo
1000
>>> img.GetCols()                                                  #Chiamata a metodo
1000
>>> img1 = ps.Image(img)                                           #Costruttore di copia
>>> img1.GetCols()                                                 #Chiamata a metodo
1000
>>> img2 = ps.Image()                                              #Costruttore di imagine vuota
>>> img2.CopyOf(img1)                                              #Assegnatore di copia
>>> img2.GetCols()                                                 #Chiamata a metodo
1000
>>> ps
<module 'pySigma' from 'D:\lavoro\iac\pySigma\Release\vs2005\pySigma.pyc'>
>>> img
<pySigma.Image; proxy of <Swig Object of type 'Image *' at 0x4a953f0> >
```

Figura 3-8 Esempio di utilizzo della classe Image in Python

Per ottenere questo risultato è necessario scrivere un file di interfaccia, pySigma.i, che si prenda cura della generazione della stessa. La scrittura di questo file è, in generale, decisamente semplice ed avviene solo una volta. Le aggiunte o le eliminazioni di proprietà e metodi non comportano, di norma, la modifica del file di interfaccia, in quanto questo contiene al suo interno tutti gli header delle classi della libreria ed è in grado di rilevare da solo le modifiche.

Per la scrittura del file di interfaccia si fa riferimento al manuale di SWIG al sito http://www.swig.org/Doc1.1/HTML/Contents.html e allegato al presente documento.

Il file di interfaccia così ottenuto mette in comunicazione Python e C++, ma non permette un efficace passaggio di dati in input e in output. L'obiettivo della implementazione qui proposta consiste nel consentire l'utilizzo, in ambiente Python, di array derivati dalla libreria scientifica numpy, lasciando in C++ i tipici puntatori. Questo consente, come descritto nella parte dedicata alla visualizzazione e alla interattività, di collegare le librerie di visualizzazione enthought.tvtk direttamente alle classi pySigma, mediate oggetti numpy.ndarray. Per una descrizione accurata della libreria numpy si fa riferimento al sito http://numpy.scipy.org/ e al manuale allegato al presente lavoro.

A questo scopo occorre introdurre, in pySigma.i, il file di interfaccia numpy.i, che contiene le definizioni di un discreto numero di %typemaps, rendendo accessibile il meccanismo di passaggio diretto di oggetti numpy.ndarray. Come si può vedere dall'esempio in Figura 3-9, l'array numpy può essere passato come argomento al metodo SetImage di *img* senza specificarne le dimensioni, in quanto sarà l'interfaccia

numpy.i a generare un wrapper in grado di leggere questi dati direttamente dalla struttura dell'array *data*.

Per una descrizione dettagliata dell'utilizzo di numpy.i si fa riferimento al documento numpy_swig.pdf allegato al presente lavoro.

Si fa notare che l'utilizzo di oggetti istanziati da classi della libreria pySigma come argomenti di metodi non comporta alcuna modifica del file di interfaccia. Un esempio di tale utilizzo è riportato nelle ultime righe di Figura 3-9.

```
>>> import numpy
>>> data=numpy.ndarray((1000,500),dtype=numpy.float32)
>>> img = ps.Image()
>>> img.SetImage(data)
0
>>> img.GetCols()
500
>>> img.GetRows()
1000
>>> rgb = ps.RGBImage()
>>> rgb.SetImage(img,img,img)
>>> rgb.GetCols()
500
>>> rgb.GetRows()
1000
>>>
```

Figura 3-9 Assegnazione di valori a un oggetto Image mediante array numpy (dato pySigma <float32>), assegnazione di valori all'oggetto RGGImage

4 L'ambiente Python

Lo schema generale dell'ambiente Python progettato per pySigma è descritto in Figura 4-1.

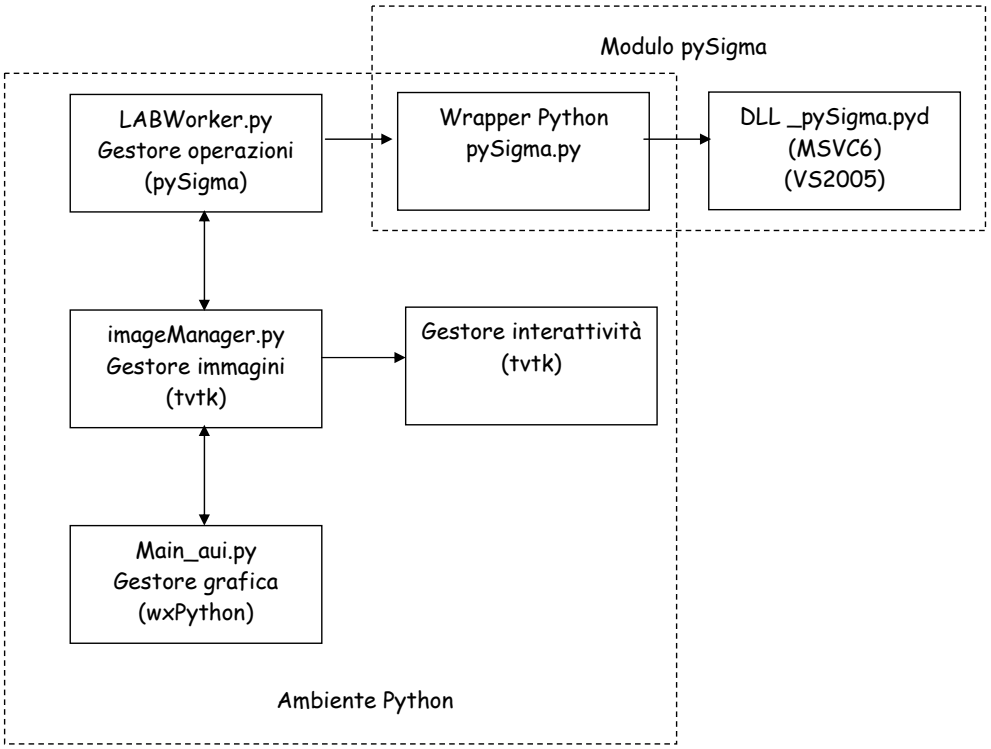


Figura 4-1 Schema generale dell'ambiente Python per la applicazione pySigma

4.1 Il gestore della grafica

Il modulo principale, main_aui.py, si basa sul il modulo wx.aui (Advanced User Interface), che espone metodi atti a generare e gestire applicazioni che presentano una finestra principale (parent), all'interno della quale compaiono sotto finestre (children), cui sono delegate specifiche funzionalità.

Da un punto di vista dell'utente, queste finestre sono trascinabili all'interno della finestra principale, sono espandibili e hanno proprietà modificabili a piacere.

Da un punto di vista dello sviluppatore, queste finestre vengono generate all'avvio della applicazione, raggruppate in "blocchi d'uso" concettuale e richiamate quando l'utente seleziona una determinata modalità di lavoro. I blocchi d'uso concettuali prendono il nome di "perspectives" che, in questa applicazione, sono definite nel modulo main_aui.py e selezionate dai tasti della toolbar.

In pratica, si generano le sotto finestre, istanziando la classe che le rappresenta sotto il controllo del manager di aui, poi si raggruppano le stesse in diverse

"perspectives", attraverso lo stesso manager di perspectives messo a disposizione da wx.aui.

Le perspectives dichiarate per la presente applicazione sono:

- "Default Startup": configurazione delle finestre all'avvio, espone all'utente il menu, la toolbar e il visualizzatore con l'immagine di presentazione
- "CIE L*A*B*": espone il pannello per l'analisi nello spazio CIE L*A*B*
- "L*A*B* histograms": aggiunge al precedente il pannello con gli istogrammi LAB

e sono riportate in Figura 4-2, Figura 4-3 e Figura 4-4.

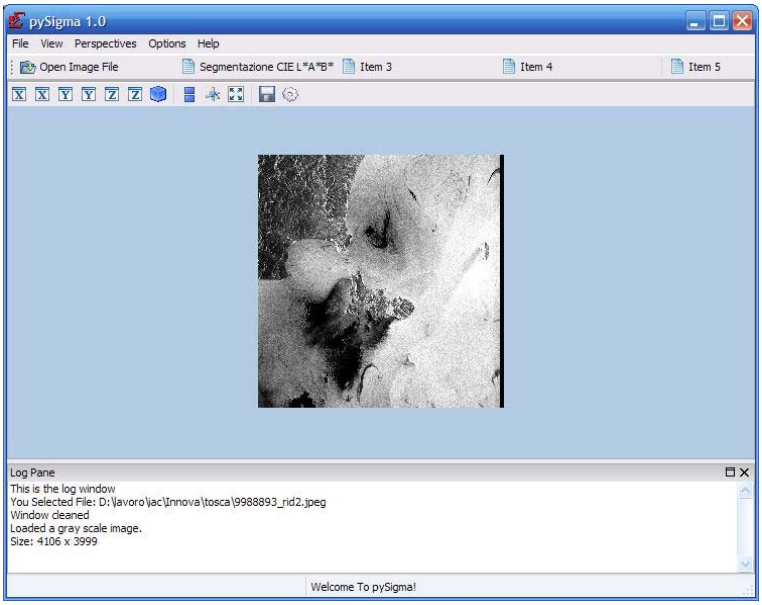


Figura 4-2 Perspective "Default Startup"

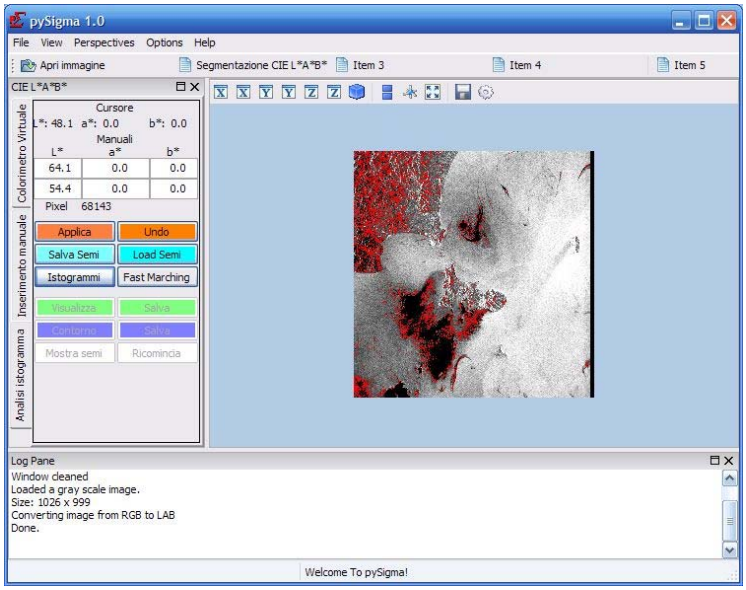


Figura 4-3 Perspective "CIE L*A*B*"

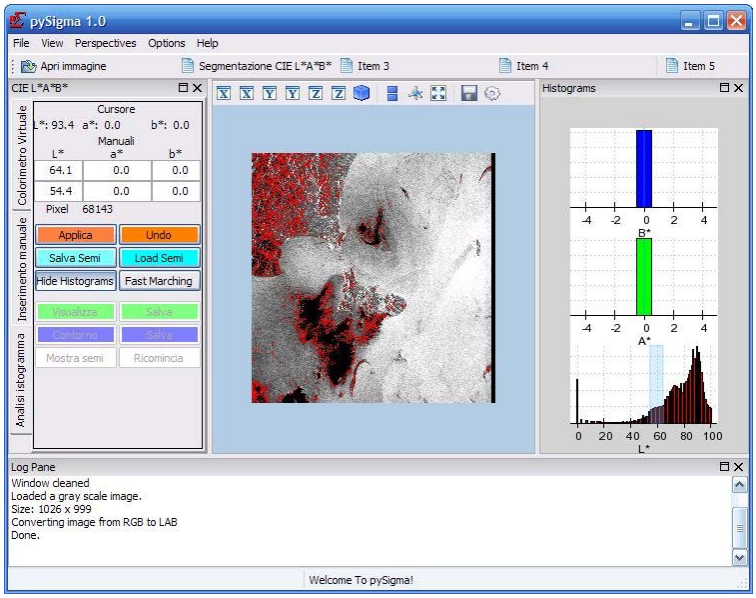


Figura 4-4 Perspective "L*A*B* histograms":

Il risultato finale è riportato all'interno di una perspective "CIE L*A*B*". Vengono attivati i comandi per elaborare il risultato e disattivati quelli per definire le soglie sull'istogramma (Figura 4-5).

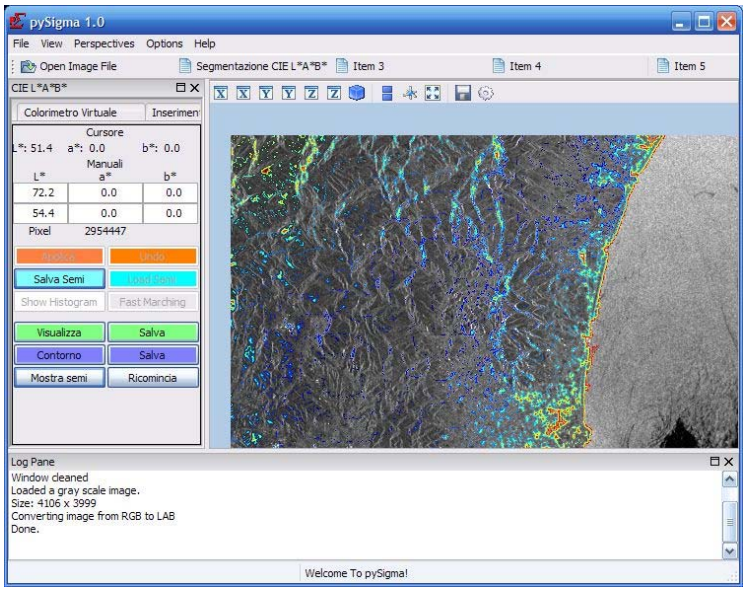


Figura 4-5 Risultato in una perspective "CIE L*A*B*"

Le perspectives sono configurate nel modulo main_aui.py, classe PyAUIFrame, metodo createPerspectives(self).

Le finestre e i menu utilizzati sono istanziati precedentemente nel modulo main_aui.py, classe PyAUIFrame, metodo __init__(self).

I metodi per istanziare le finestre sono riportati nel seguito.

PyAUIFrame.CreateIAC_DecoratedSceneCtrl: istanzia un oggetto IAC_Panel (modulo IAC_Decorated_Scene.py) come istanza derivata da enthought.pyface.tvtk.decorated_scene. Quest'ultima è una classe contenuta nel

modulo enthought.pyface.tvtk.decorated_scene, basata sulla classe wxVTKRenderWindowInteractor, che espone proprietà di visualizzazione e metodi di interazione estremamente efficaci e completamente configurabili. Questa classe aggiunge, alle originali proprietà grafiche del renderWindow VTK, una toolbar che permette di modificare i punti di vista, accedere alla modalità full-screen, salvare la scena visualizzata e infine configurarne le proprietà attraverso le modalità tipiche dei Traits. La classe derivata IAC_panel, oltre a generare l'immagine iniziale della applicazione, aggiunge alla classe base due osservatori (metodo "AddObserver") che generano un evento a seguito delle azioni "MouseMoveEvent" e "LeftButtonReleaseEvent". Il primo evento è catturato per visualizzare la posizione del mouse e i valori delle triplette RGB e LAB, il secondo è catturato per definire la posizione dei semi all'interno dell'immagine. I valori della tripletta RGB sono riportati sulla status bar della finestra principale, in basso a sinistra (Figura 4-4). I valori della tripletta LAB sono riportati nel pannello di sinistra, sotto la label "Cursore" (Figura 4-4).

PyAUIFrame.CreateIAC_ChacoCtrl: istanzia un oggetto IAC_ChacoPanel, costruito con gli strumenti esposti dal modulo enthought.chaco2. Questo oggetto è costituito da tre grafici a barre, disposti in verticale, che riportano gli istogrammi LAB dell'immagine selezionata, dopo che questa è stata convertita nello spazio di colore LAB. A questi istogrammi viene sovrapposta dinamicamente la selezione effettuata dall'utente attraverso la scelta dei semi sull'immagine (Figura 4-4).

PyAUIFrame.CreateTextCtrl: istanzia una finestra di testo, comune a tutte le perspectives, che viene utilizzata come finestra di log per riportare all'utente il risultato di alcune azioni e il procedere delle stesse.

4.2 Il gestore delle immagini

Il gestore delle immagini è contenuto nel modulo imageManager.py ed è rappresentato dalla classe ImageManager nel modulo imageManager.py.

Questo oggetto mette a disposizione un browser di immagine per la selezione del file di input (vedi Figura 4-6), carica il dato e costruisce una pipeline di vtk avente le caratteristiche riportate in Figura 4-7.

La mask viene aggiornata ogni volta che cambia la selezione dei minimi e dei massimi per l'immagine LAB, ovvero quando un seme selezionato dall'utente viene inserito nella lista dei semi. Il modulo della gestione delle attività comanda il cambiamento di mask nel metodo UpdateLab.

L'oggetto tvtk.ImageMask fornisce i dati all'oggetto tvtk.ImageActor che viene visualizzato nella scena (variabile ia).

Una volta effettuato il calcolo, l'attore originale viene nascosto, mentre viene visualizzato un nuovo attore (iaresult) definito nel metodo replaceActor. Quest'ultimo

metodo richiede a parametro l'array di numpy ottenuto attraverso il modulo pySigma.RGBImage.GetWholeImg.

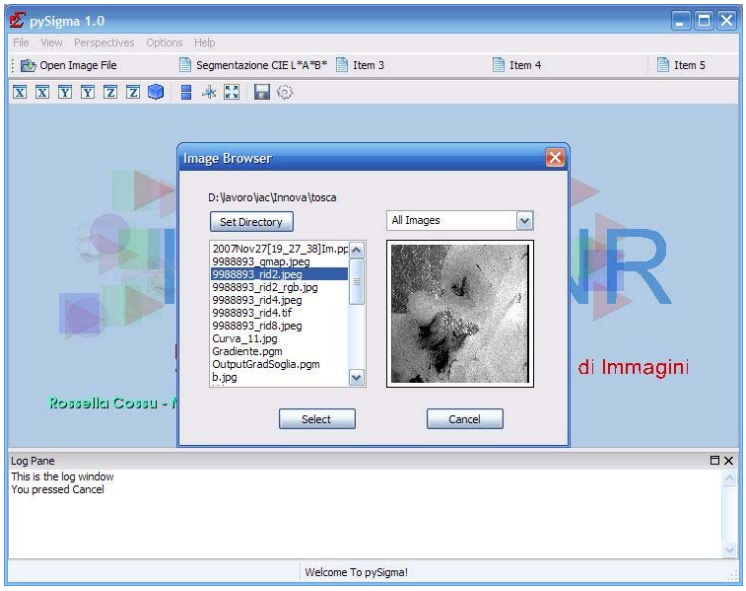


Figura 4-6 Image Browser per la selezione del file di input.

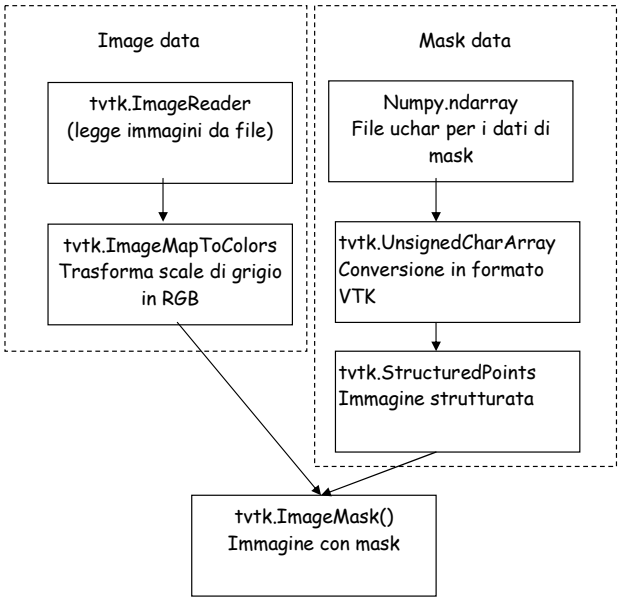


Figura 4-7 Oggetto da tvtk.ImageMask ottenuto sovrapponendo l'immagine originale e la mask derivata dalle soglie impostate dall'utente nello spazio LAB.

4.3 Il gestore delle interattività

Il gestore delle interattività è ottenuto come classe che ha, tra le sue proprietà, l'istanza di enthought.pyface.tvtk.decorated_scene. Questa classe mette a disposizione una scena di VTK (wxVTKRenderWindowInteractor) a cui è aggiunta una toolbar che gestisce i punti di vista, il full-screen e il salvataggio della scena stessa.

A questa scena viene aggiunto un metodo che inserisce un picker e due observers, in modo da poter catturare gli eventi "MouseMoveEvent" e "LeftButtonReleaseEvent", da cui vengono estratti i valori RGB e LAB utili alle elaborazioni successive. Il picker trasforma le coordinate del mouse in coordinate nello spazio dell'immagine.

4.4 Il gestore delle operazioni

Il gestore delle operazioni viene istanziato a seguito della scelta dell'utente sul tipo di calcolo da effettuare sull'immagine. Attualmente l'unico gestore disponibile è relativo alla segmentazione attraverso l'analisi dell'istogramma, nella modalità FM Soglia.

Il gestore delle operazioni è denominato internamente activeWorker, ed è istanziato nella classe PyAUIFrame. Esso riceve in input, al momento della creazione, un puntatore al gestore di immagini, uno al gestore di interattività e uno al pannello dei comandi che lo riguardano. In questo modo il gestore di operazioni è in grado di interagire con tutti gli altri gestori, per recuperare dati e parametri ed assegnare il risultato.

5 Inserimento di nuove funzionalità in pySigma

5.1 Inserimento di una classe c++

Per aggiungere una classe C++ da esporre nell'ambiente python, occorre inserire la classe nel progetto VS2005 o MSVC6 e includere l'header nel file pySigma.i. Vanno evitati metodi che restituiscono array o vettori, in quanto non è possibile definirne le dimensioni dall'ambiente Python. Se è necessario importare o esportare vettori o matrici occorre farlo attraverso la lista dei parametri e le relative interfacce vanno dichiarate attraverso le %>typemap messe a disposizione da numpi.i.

5.2 Inserimento di una attività

Per aggiungere una attività occorre definire un gestore di attività e un pannello dei comandi.

Il pannello dei comandi può essere aggiunto al notebook attualmente presente per le analisi nello spazio LAB. Questo notebook è definito nel modulo MultipadPanel.py. In ogni pagina del notebook può essere definito un pannello dei comandi, realizzato sullo schema del pannello attualmente presente (modulo Analisi_istogramma_panel.py). L'uso dell'editor grafico BOA semplifica notevolmente la realizzazione di queste piccole interfaccia. Il notebook con i controlli è generato in PyAUIFrame, nel metodo "CreateMultipadCtrl". Esso è associato alla perspective "CIE L*A*B*".

Nel caso in cui si voglia utilizzare un pannello in una nuova perspective, occorre definire un nuovo tipo di controllo, attraverso un nuovo metodo basato sullo schema "CreateMultipadCtrl", da inserire nella classe PyAUIFrame e da richiamare nel suo metodo __init__. La nuova perspective andrà associata a uno dei tasti della toolbar e da esso richiamata.

L'aggiunta di una attività dovrebbe comportare sempre la definizione di un nuovo gestore delle attività. Per operare nello spazio LAB, si consiglia di effettuare la copia del gestore presente workerLAB.py ed effettuare le modifiche su di esso.

Il riutilizzo di frammenti di codice preesistenti consente la modifica delle azioni e la replica delle funzionalità già inserite nel codice.

6 Manuale d'uso

Dopo aver lanciato il codice dall'eseguibile main_aui.exe o da finestra dos con il comando pythonw main_aui.py, viene visualizzata la finestra riportata nella copertina.

Cliccare sul bottone della toolbar "Apri immagine" per accedere a un browser di immagini che consente la selezione del file da analizzare. L'immagine viene caricata nell'ambiente pySigma e viene resa disponibile una modalità interattiva 2D, che consente di effettuare zoom trascinando il mouse con il pulsante destro premuto o pan dell'immagine trascinando il mouse con il pulsante centrale premuto. Zoom in e out possono essere effettuati, dopo aver selezionato l'immagine (click con pulsante sinistro) mediante i tasti <+> e <->. La modalità a schermo intero viene attivata dalla toolbar interna alla scena.

Cliccare sul bottone della toolbar "Segmentazione CIEL*A*B*" per effettuare la conversione dell'immagine prescelta nello spazio LAB e attivare il relativo pannello comandi (Figura 4-3).

La selezione dei semi sull'immagine determina i valori LAB minimi e massimi su cui verrà effettuata la soglia dell'immagine. I semi vengono selezionati con un click del tasto sinistro del mouse. Una selezione errata può essere annullata mediante il tasto "Undo". I semi selezionati possono essere salvati cliccando sul tasto "Salva semi". I semi salvati possono essere ricaricati cliccando sul tasto "Load semi". Il tasto "Istogrammi" visualizza il pannello degli istogrammi. L'intervallo nello spazio LAB selezionato attraverso la scelta dei semi è evidenziato sugli istogrammi stessi (Figura 4-4).

L'analisi dell'immagine viene attivata cliccando sul bottone "Fast Marching", che apre una finestra di dialogo in cui possono essere inseriti i parametri per l'analisi.

Completato il calcolo, viene visualizzata l'immagine risultato, con un contorno per ogni istante temporaLe selezionato a parametro.

Il tasto "Contorno" permette di selezionare una singola curva ad uno specifico istante temporale. I tasti "Salva" salvano l'immagine visualizzata nella scena in un formato tif.

Il tasto "Mostra semi" visualizza i semi prescelti per l'analisi.

Il procedimento può essere riavviato attraverso il tasto "Ricomincia".