國立交通大學

資訊科學與工程研究所

碩 士 論 文

共軛焦顯微鏡影像分析系統之設計與實作

Design and Implementation of Software Systems for Analyzing Confocal Microscopy Images

研 究 生：潘笙煊

指導教授：荊宇泰　教授

中 華 民 國　九 十 五　年 七 月

共軛焦顯微鏡影像分析系統之設計與實作

Design and Implementation of Software Systems for Analyzing Confocal Microscopy Images

研 究 生：潘笙煊　　Student：Da-Ming Chang

指導教授：荊宇泰　　Advisor：Yu-Tai Ching

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

國立交通大學

研究所碩士班

論文口試委員會審定書

本校 資訊科學與工程 研究所 潘笹煊 君

所提論文:

共軛焦顯微鏡影像分析系統之設計與實作

合於碩士資格水準、業經本委員會評審認可。

口試委員：許勝雄 _____

胡敏君 _____

指導教授：荊宇泰

所　　長：[illegible]

中 華 民 國 九十五 年 7 月 14 日

共軛焦顯微鏡影像分析系統之設計與實作

學生：潘笙煊　　指導教授：荊宇泰 博士

國立交通大學資訊科學與工程研究所

摘要

在生物、醫學研究領域上，常常需要在某些實驗影像當中尋找重要的特徵。儘管當前已有許多優良的影像分析與辨識技術，仍有部分影像處理、特徵篩選的工作需要經由人工以互動式的方式操作電腦軟體來完成。有鑑於此，本論文將提出一種以視窗介面為基礎，專門分析與處理共軛焦顯微鏡影像之 C++程式開發架構。我們以 Trolltech 公司所研發的跨平台視窗程式類別庫(class library)－Qt，作為我們實作的平台。本專案之實作將便利共軛焦顯微鏡影像之分析工作。

關鍵字：系統分析、系統設計、設計模式

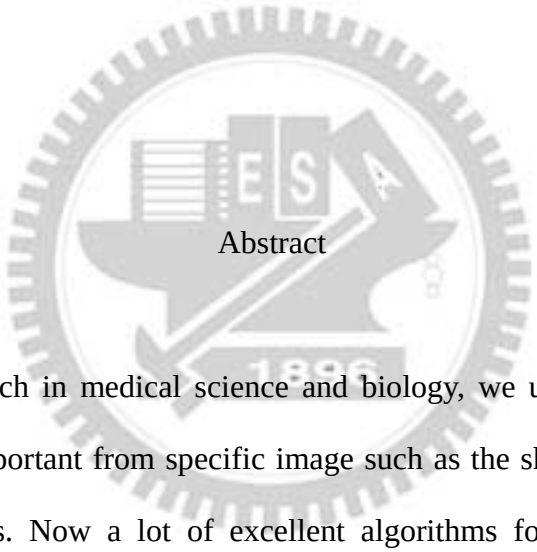
Design and Implementation of Software Systems

for Analyzing Confocal Microscopy Images

Student: Ren-Shiuan Pan Advisor: Dr. Yu-Tai Ching

Institute of Computer Science

National Chiao Tung University



Abstract

When we research in medical science and biology, we usually have to obtain some information important from specific image such as the shapes and locations of neurons in fly brains. Now a lot of excellent algorithms for image analysis and processing are available, however, some of such work still needs to be done manually or interactively. Therefore, we propose a user-interface system for analyzing and processing biological/medical images from confocal microscopy. The system is cross-platform and implemented with Trolltech® Qt Library, based on C++ programming language.

Keyword: System Analysis, System Design, Design Patterns

誌謝

兩年來，我從這裡找到了自信。還記得當初進入實驗室，指導教授對我說：「進入我們實驗室，就不要怕寫程式。」這對於執著於程式實作的我而言，實在是一則令人喜悅的好消息。然而，在參與本實驗室所進行計畫與論文的程式實作過程當中，我漸漸的發現自己專業能力與心態上的不足之處。

因此在這裡，我要感謝我的指導老師，**荊宇泰**教授。感謝您在這兩年來用心的指引我研究方向與處事態度，並針對我的興趣與專長，為我指定了最合適的研究題目，讓我得以自由的發揮所學所用，使盡渾身解數致力於此。

感謝口試委員**謝昌煥**博士與**胡毓志**教授，在口試當天遇到颱風，大雨直下的天氣狀況仍前來為我口試，並針對我當天報告的不足之處提出問題。感謝謝昌煥博士您的耐心，為我的論文細心地提出指正與建議。

致同時為室友與同學的昆世，感謝你每次好心借我機車和你順路送來晚餐讓我少了許多後顧之憂。感謝實驗室同學俊瑋，偉谷和偉志，在我學習的路上遇到困難時總是和我有說有笑，讓我暫時忘了不愉快並重新出發。感謝秉璋耐心解說提供給我的論文和程式碼，與林志陽博士提供的程式實作，讓我的專案進行的更為順利。

最後感謝在台北的家人適時的為我貼補生活費，與不定期的電話關心，讓我不必擔心民生問題，能夠擁有更多時間專心致力於學業上。感謝母親蕙臻女士每逢我回台北之時為我準備營養又可口的晚餐，讓我的身體健康總是得到適時的照顧。感謝女友汶婷小姐，儘管台北－新竹兩地之隔，仍每天以電話聯絡，給予我精神上的支持，讓我得以順利完成本論文之實作。

目錄

圖目錄

第一章　緒論

1.1 問題背景

在生物和醫學的研究上，生物體某些特殊部位的細胞數目、形狀、大小與位置在研究生物的行為、學習模式與疾病上扮演著重要的角色。如今，由清華大學江安世教授所指導的團隊，在基因轉殖之老年癡呆變種果蠅腦中觀察到空洞化的現象。空洞的數目與分布情形可以用來幫助判斷果蠅的疾病發展的過程。這樣的生物特徵（例如果蠅腦），通常需要經由高精密度的共軛焦顯微鏡（註，於第 3 頁）取得影像，並經由電腦影像處理、圖形辨識技術從事特徵搜尋、分析、比對等工作。本實驗室已研發出許多演算法，得以實現共軛焦顯微鏡影像分析、特徵搜尋、幾何結構重建等工作，例如果蠅腦細胞之計數[1]、退化的腦細胞之搜尋與體積計算[2]、果蠅大腦之 3D 幾何結構重建[3]均已有相當良好的表現。

儘管當前已有良好的影像分析演算法，仍時常需要以人工點選的方式完成影像局部選取、參數設定、演算法執行結果的修正等動作。部分的工作或許還能夠依靠目前市面上的影像處理軟體來完成。這類套裝軟體通常對於常見之二維影像格式較為支援，且所提供的功能還是無法對於某特定應用需求，例如共軛焦顯微鏡影像的存取與操作，而做出特別的設計，因此還是需要自行撰寫使用者介面程式，經由單機(Single Alone)的應用程式，或是透過 Web Service 的方式操作，來便利影像處理的人工作業。

1.2 研究問題

當前已有許多使用者介面開發工具。諸如 Microsoft 的 MFC、Win32 SDK、Borland C++ Builder 的 OWL、跨平台的 wxWidgets、GTK 與 Trolltech 的 Qt 等。善用上述工具，得以讓分析與處理共軛焦顯微鏡影像的程式實作能夠使用者介面化，好讓操作演算法的使用者能夠立即看見實驗的結果，以及輕易的對其完成人

工處理的動作。然而，這樣的程式實作常常因為相依於上述之開發工具，並且因為規模較小，設計時沒有一定的準則，造成修改實作與延伸功能的工作變得相當困難，尤其是在面對演算法的程式實作需要搬移至別的作業系統上面執行之時，有時重新撰寫系統仍比修改來得容易。

1.3 研究動機與目的

有鑑於此，我們這時不禁有個念頭：砍樹之前，得先將斧頭磨至銳利。因此，我們是否能夠以軟體工程的角度，提出一種 C++程式開發架構，並以此架構設計出一個視窗式介面軟體，使得共軛焦顯微鏡影像處理的演算法、應用需求，與使用者介面互動系統，均能以清楚、一致的方式順利實作。於是形成了研究此問題的動機。

本專案分為三個階段實作完成，並提供兩種不同應用需求作為範例。在第一階段裡，我們將實作一組作用於影像基本操作的類別函式庫(class library)。在第二階段當中，我們經由連接至第一階段所述之函式庫，設計出屬於某種應用需求（例如，NeuronTracing）之功能模型，作為使用者介面程式與影像處理演算法實作之間的系統介面。第三階段則是以 Qt 設計出視窗式使用者介面，經由操作第二階段之功能模型得以完成該需求之影像分析工作。範例中的兩種應用需求分別為 1. 果蠅腦之老年痴呆情形分析[1]、2. 半自動果蠅腦神經追蹤之使用者介面。

1.4 論文架構

本論文共六章，第一章介紹我們的研究動機與研究問題，第二章介紹本論文包括應用需求範例之相關研究。第三章介紹兩應用需求之需求分析、與本專案之系統分析，第四章介紹本系統在影像處理與應用需求上的系統設計，第五章則介紹使用者介面系統的設計與展示，第六章則為本論文做總結與討論未來計畫。

註－顯微鏡取像技術 (節錄自[2])

在本系統中所作用的影像，資料來源均從**共軛焦顯微鏡**取得。我們在此，先大致介紹一些較基礎的顯微鏡取像方法和它的限制。其中包括：光學顯微鏡(Light Microscopy)、螢光顯微鏡(Florescence Microscopy)與共軛焦顯微鏡(Confocal Microscopy)等三種取像方式。

I. 光學顯微鏡

一般來說，光學顯微鏡利用可見光中不被目標物體所吸收的光來成像，圖 1.1 左側為光學顯微鏡的示意圖，光從光源利用一個透鏡聚焦在目標物體上，能通過目標物體的光再經過兩個透鏡的放大和聚焦作用成像在偵測器上，利用光學顯微鏡來取像的影像其解晰度受限於光源的波長為 0.2μm，也就是藍光波長的一半。

II. 螢光顯微鏡

螢光顯微鏡不同於光學顯微鏡，它利用螢光分子吸收特定波長的光並放射出較長波長的光來成像，圖 1.1 中間的圖為螢光顯微鏡的示意圖，光從光源經過一個濾波鏡後經由雙色向面鏡聚焦到目標物體上，目標物體上的螢光分子吸收光線後放出特定波長的光再穿過雙色向面鏡，之後會再經過另一個濾波鏡然後才聚焦在偵測器上，由於螢光顯微鏡的光源一樣為可見光，所以所取的影像解析度一樣受限在 0.2μm。

III. 共軛焦顯微鏡

共軛焦顯微鏡成像的方式約略跟螢光顯微鏡類似，是利用螢光分子放射出來的光線來成像，不同點在於光源的選擇，共軛焦顯微鏡利用的是擁有好的聚焦性的雷射來當作光源，圖 1.1 右側與圖 1.2 是共軛焦顯微鏡

的示意圖，雷射光線經由雙色向面鏡後聚焦在目標物體上，物體裡的螢光分子吸收光線後再放射出其他波長的光線，放射出的光線經過雙色向面鏡後再經過一個針孔之後成像在偵測器上，由於使用光源不同，共軛焦顯微鏡的解析度比起前述兩種顯微鏡會好很多。

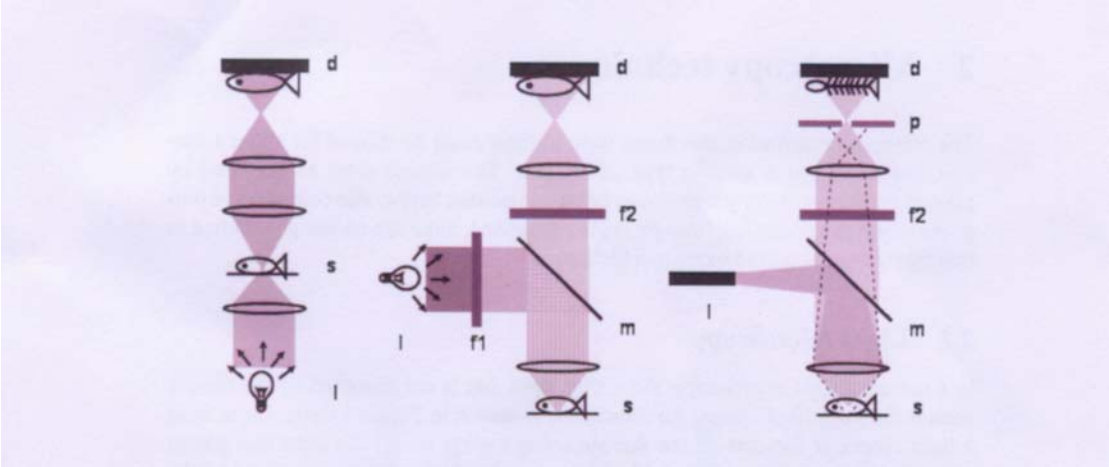


圖 1.1： 光學顯微鏡(左)，螢光顯微鏡(中)，共軛焦顯微鏡(右)。所有顯微鏡都擁有一個光源(I)，一個樣本(s)，和一個偵測器(d)。除此之外螢光顯微鏡還有兩個濾波鏡(f1，f2)和一個雙色向面鏡(m)，而共軛焦顯微鏡則有一個濾波鏡(f2)，一個雙色向面鏡(m)和一個針孔(p)。

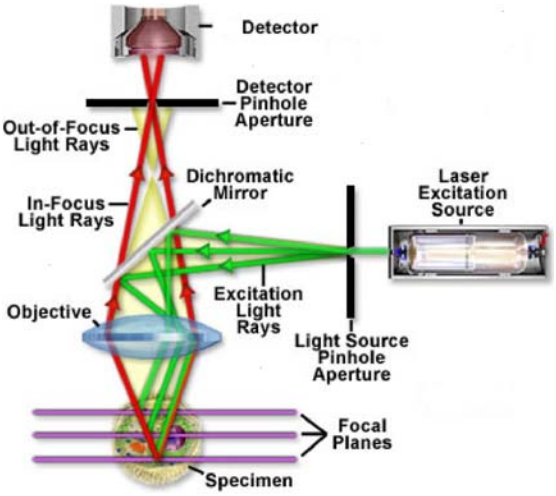


圖 1.2 共軛焦顯微鏡取像示意圖

第二章 相關研究

2.1 果蠅腦之老年痴呆情形分析

果蠅的腦細胞在高齡期時會出現腦室(vacuoles)，也就是老年痴呆現象。國立交通大學，醫學影像實驗室的李秉璋(P. C. Lee)同學提出了一種方法[1]，用來找出果蠅腦之三維影像當中所有的 vacuoles 之位置以及體積計算，下頁是此演算法的圖示範例。圖 2.1(a)是原始影像，他首先以 Gaussian Matched Filter 於各張果蠅腦影像進行圓形特徵的搜尋，以 512x512 的影像為例，半徑以 5-7 個 pixel 為佳。如圖 2.1(b)，完成 Matched Filter 以後，對影像進行二分法，得出亮的部分即為 vacuoles 所在位置。接著，對這些亮的區塊以 Delaunay Triangulation 找出某個特定參數的 alpha-shape，如圖 2.1(c)。完成之後會得到一組 contours(輪廓資料)，將這些 contours 進行 snake (active contour)演算法，即可確認果蠅腦各層影像之 vacuoles 的位置與確切形狀，如圖 2.1(d)。圖 2.1(e)與圖 2.1(f)為代入 snake 演算法之前與之後的比較，顯然執行 snake 以後的輪廓更貼近為原 vacuoles 的邊界，使得總體積能夠更為精確計算。

然而找出的 vacuoles 當中，有些也有可能是雜訊[1]，或是其他部位的相似特徵，例如食道這個部位，中間也成圓柱狀空洞，因此也會被當作 vacuoles 找出。在這樣的情況，必須以人工方式刪除電腦誤判的 contours，再重新計算 vacuoles 的總體積。此時，若沒有一組人性化的使用者介面，儘管輪廓已經標示出來，仍然需要輸出各 contours 的點座標以人工方式比對並刪除錯誤的訊號。這會是相當費時費力的工作，因此我們以此案例作為本專案的應用需求範例之一。

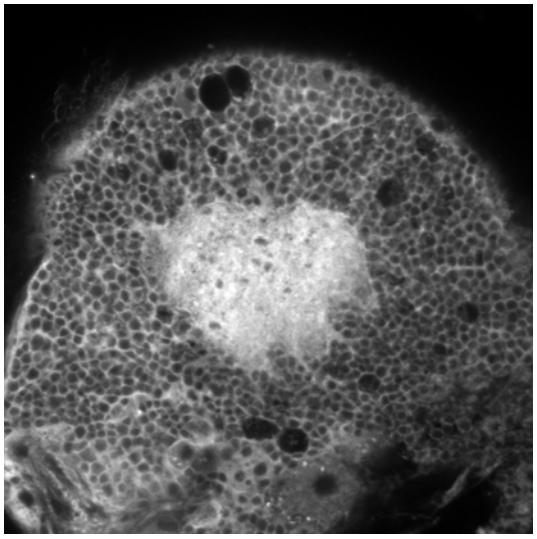
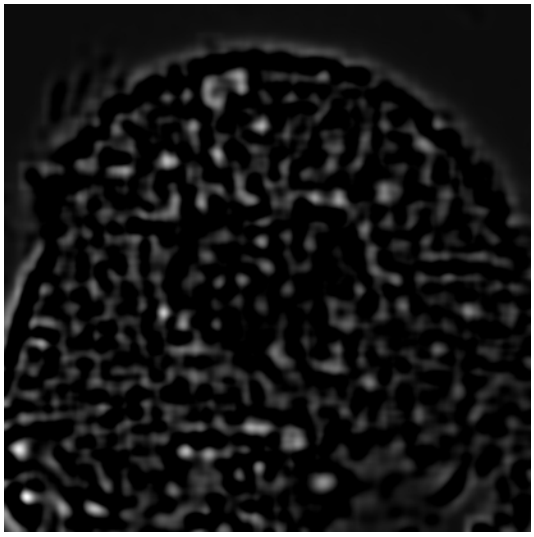



圖2.1(a) 果蠅腦之老年痴呆情形分析演算法圖示範例 (b)

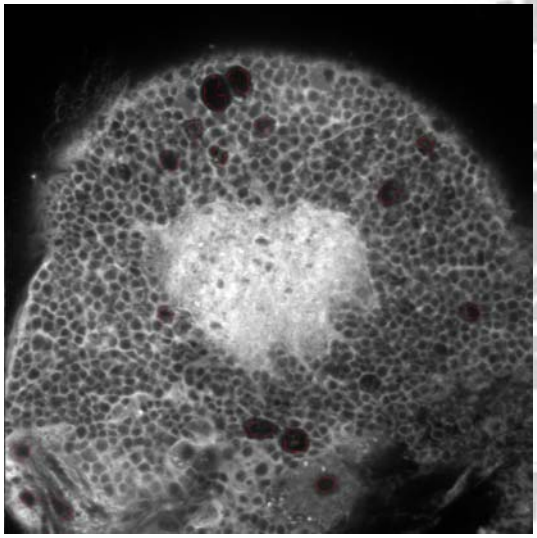


圖2.1 (c)

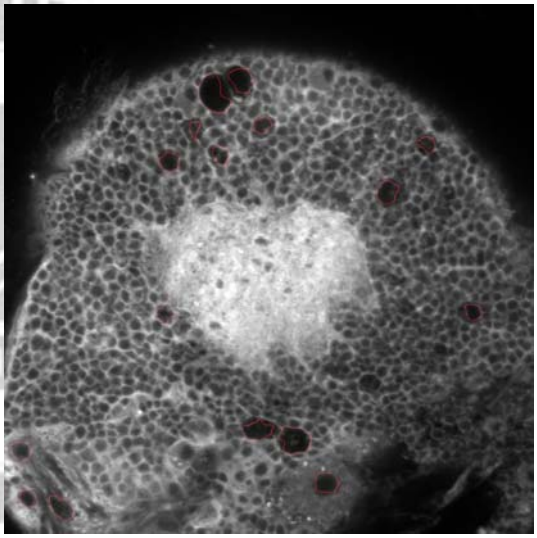


圖2.1 (d)

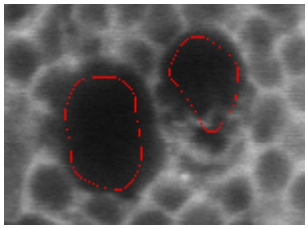


圖 2.1 (e)

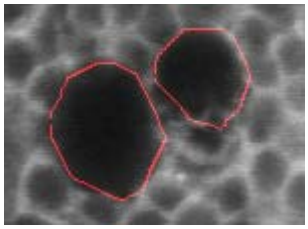


圖 2.1 (f)

2.2 Active Contour 與 GVF 之相關研究

2.2.1 Active Contour Method [4]

Active-Contour Method 又稱為 Snake Method[4]，是一種將連續的曲線做能量最小化的可變形模型。藉由定義 snake 的內力(internal forces)與外力(external forces)讓 snake 的輪廓曲線能夠變形至影像中的物體邊界或是期望的影像特徵上。Snake 被廣泛的運用在各種應用程式上，如邊界的偵測、影像的模型重建、影像分割及行動追蹤上。傳統上，在影像中移動的 snake 曲線為 $v(s)=[x(s),y(s)], s\in[0,1]$, s 是一個長度參數，而能量函數定義為

$$E=\int_0^1 \frac{1}{2}[\alpha|v'(s)|^2+\beta|v''(s)|^2]+E_{ext}(v(s))ds \qquad (1)$$

其中α與β是用來控制active contour的權重參數，分別為連續性(continuity)與平滑性(smoothness)的強度。另外，$v'(s)$ 與 $v''(s)$ 分別為 $v(s)$ 對於s的一階與二階微分。其中 E_{ext} 是snake的外在能量函數，則是從影像當中得來。我們通常對其定義較小的值於影像當中較為感興趣的特徵上面，例如某個物體的邊界等等。

就一張灰階的影像I(x, y)而言（I(x, y)表示，位於影像座標x, y的像素之亮度），我們可將這張影像的像素視為連續的函數I(x, y)，典型的外在能量設計希望snake上的點能夠收斂在物體的邊界，可定義如下：

$$E_{ext}^{(1)}(x,y)=-|\nabla I(x,y)|^2 \qquad (2)$$

$$E_{ext}^{(2)}(x,y)=-|\nabla[G_\sigma(x,y)*I(x,y)]|^2 \qquad (3)$$

其中 $G_\sigma(x,y)$ 是一個二維的標準差為 σ 的高斯函數而 ∇ 則是梯度運算子(gradient operator)，從(3)的外在能量定義可以簡單的看出當 σ 越大時，將使得邊界變得更模糊。然而，當需要讓active contour在影像中受到較大的範圍影響，則需要較大的 σ 值。

對snake做能量最小化必須滿足Eular Equation

$$\alpha x''(s)-\beta x^{(4)}(s)-\nabla E_{ext}=0 \qquad (4)$$

可將上式當作靜力平衡等式

$$F_{\text{int}} + F_{ext}^{(p)} = 0 \qquad (5)$$

其中 $F_{\text{int}} = \alpha x''(s) - \beta x^{(4)}(s) - \nabla E_{ext}$ 且 $F_{ext}^{(p)} = -\nabla E_{ext}$ 。內力 F_{int} 防止snake的過度延伸與彎曲，而外力 $F_{ext}^{(p)}$ 將snake的輪廓曲線推向期望的影像邊界。

2.2.2 Gradient Vector Flow [5][6]

Gradient Vector Flow 是一種以PDE (Partial Differential Equations，部分微分方程) 為基礎的向量擴散方法。GVF常作為輔助snake model強化影像當中的邊界搜尋與影像分割之準確度。[5]所提到的主要方法，是以PDE來產生一組新的external force field(相較2.2.1所提的影像本身作為外力的依據)，使得snake model受到較少因為雜訊而形成的外力(external force)干擾，便能正確的貼合於影像的特徵之邊界處。

在此我們以圖示的方式介紹這樣的方法(圖片取自[6])：

圖2.2(a)是原始影像，圖2.2(b)是原始影像的GVF，綠色的箭頭表示該像素的GVF之向量，越長表示該點的梯度(gradient)越強。我們可以發現在影像邊界的周圍的箭頭較為明顯，且那些箭頭的方向均指向邊界，與邊界的邊線垂直。

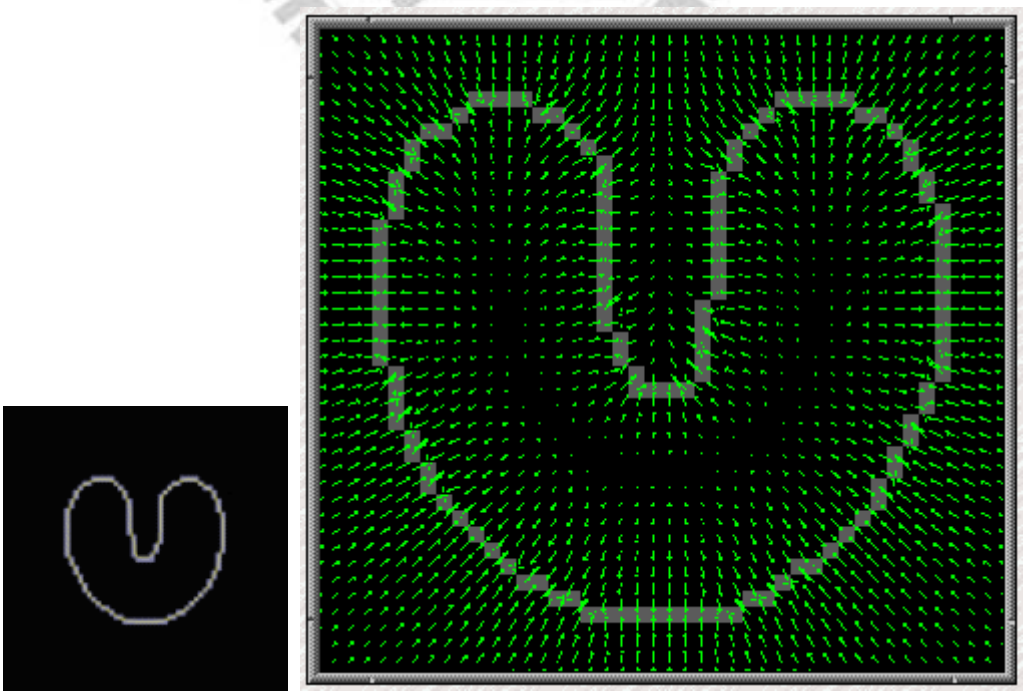


圖2.2(a) (b)

	Initial Curve	Final Curve
A traditional snake must start close to the boundary and still cannot converge to boundary concavities.		
A GVF snake can start far from the boundary and will converge to boundary concavities.		
A GVF snake can even be initialized across the boundaries, a situation that often confounds traditional snakes and balloons.		

圖2.3:以上的圖取自[6]，最上方之兩張圖說明了傳統的snake演算法對於邊界的搜尋能力的限制，中間與下方的圖形則是代用GVF演算法以後的執行結果，我們可以發現就算snake model初始狀態甚至跨越了圖形的邊界(noise較高的情形)亦能讓snake model完整附合於影像的邊界上。

2.3 相關軟體實作

2.3.1 Amira [7]

Amira 由 Mercury Computer Systems 所研發，是一套從事科學計算視覺化與三維影像之(Volume)處理、呈現(Rendering)與幾何結構重建的軟體。目前已被生物學、醫學、材料工程學、與品質管理、物理、地理學之各領域研究者廣泛使用。

Amira 除了支援一般通用的影像檔案格式，例如 ACR-NEMA/DICOM, TIFF, PPM, JPG, binary raw data, VRML, PLY, Fluent/UNS, AVS 等以外，並能夠對影像資料以自動／互動式之方式完成影像分割與特徵擷取之動作；對於 3-D 的 Volume Data，例如電子顯微鏡影像，能夠自動產生 3-D Surface Models 與 3-D Tetrahedral Models，並以 iso-surface 方式或是 direct volume rendering 方式，結合硬體支援(Graphics Hardware)配合 OpenGL 程式庫以呈現(Render)volume data。

下圖為 Amira 對於果蠅腦的 Mushroom Body 做出的 Volume Rendering：

圖 2.4 Amira 的 Volume Rendering 影像範例

除此之外，Amira 仍能提供程式開發者延伸軟體的功能，針對某種應用需求做出特別的設計，例如特定格式的影像讀取與儲存，或是自行開發的 3-D Model／Volume Rendering 之演算法，均能以 C++完成開發。此外，Amira 又提供了 Amira Development Wizard 自動產生外掛(plug-in)程式架構供開發人員能夠快速且便利的完成實作。

2.3.2 NIH Image [8]

NIH Image 是一個由 Wyane Rasbane 在美國國家衛生院(National Institutes of Health)的國家心理衛生組織(National Institute of Mental Health)所發展的科學影像處理軟體，在生物／醫學研究領域上可用來從事細胞或胞器之面積、數量、長短軸之計算，電泳結果之分析如核酸、蛋白質定量，以及共軛焦顯微鏡、斷層掃描影像分析與處理，均能提供便利的使用者介面供研究者完成工作。

NIH Image 於 Macintosh 作業系統運行。NIH Image 支援一般格式的檔案存取，例如 TIFF, PICT, PICS 與 MacPaint 之檔案格式，使得經由 NIH Image 處理之影像資料能夠讓各個不同的影像處理軟體所支援。NIH Image 可對於影像資料進行一般通用的影像處理功能，例如亮度／對比調整，影像資料統計與建檔(Density Profiling)，平滑／清晰、邊緣偵測，Median Filtering 與一般的 Spatial Convolution（可對於自行定義的 kernel 進行）。

NIH Image 以圖形化的使用者介面，提供使用者圈選感興趣的區域(Region of Interest)，對於圈選的影像資料加以量測（面積、長度等），並能將取得的資訊轉為純文字檔案輸出。各種不同的量測方式均已提供適當的圖形化之工具。另外也支援多種影像的輸入來源。例如掃描器或數位相機等。

除此之外，NIH Image 也提供了可程式介面，供使用者能夠自行定義影像處理的演算法，以滿足某種特定的應用需求。其中一種為類似 Pascal 程式語言的 Macro，經由這樣的 Macro 使用者可以自行設計影像處理的方式。另外 NIH Image 也提供了完整的 Pascal 程式碼(Open Source)供程式開發者能夠自由編輯與重新編譯。下圖為 NIH Image 的使用範例：

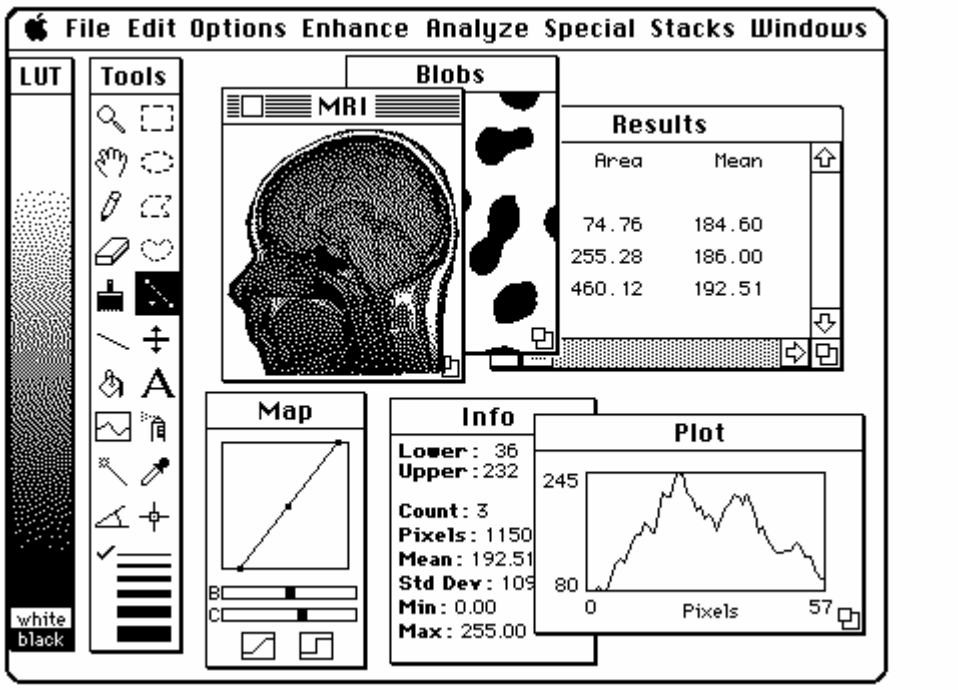


圖 2.5 NIH Image 的圖形介面示意圖 ，MRI 視窗顯示讀入的斷層掃描影像，影像經由 LUT(Look-up Table)對應色彩之後輸出至螢幕上。使用者可從 Tools 視窗選擇需要的工具，對於影像資料完成編輯、選取與測量等工作。從 Info 與 Plot 視窗可取得影像相關的統計資料。

第三章 系統分析

本專案期望提出一個以 C++為基礎的程式架構，並以此架構實作出一個視窗應用軟體，專門對於共軛焦顯微鏡影像，在影像資料（二維與三維）輸入與輸出、影像處理演算法執行、資料暫存與運算結果檢視，提供便利、直覺與適當的使用者介面互動系統。這樣的架構能夠輕易的與使用者介面開發程式庫(e.g. 在本專案中，我們使用 Qt 作為視窗介面程式設計的工具)結合以實現互動式影像資料編輯等應用需求。

在本章中，3.1 節介紹本專案兩應用需求範例之需求分析(Requirement Analysis)，其中包括第一章所提到的兩個應用需求範例。3.2 節則介紹兩應用需求範例之系統分析，以 UML(Universal Modeling Language)為基礎，描述各系統之物件、動態與功能模型。

3.1 需求分析

在本節中我們將介紹兩應用需求範例，分別為**果蠅腦之老年痴呆情形分析**、**與半自動果蠅腦神經追蹤**之使用者介面的需求分析。

3.1.1 果蠅腦之老年痴呆情形分析－需求分析

如 2.1 節所述，在參考文獻[1]中，用來找出果蠅腦之三維影像當中所有的腦室(vacuoles)之位置以及體積計算的步驟如下：

1. Gaussian Matched Filter 於各張果蠅腦影像進行圓形特徵的搜尋。
2. 完成 Matched Filter 以後，對影像進行二分法，得出亮的部分即可能為 vacuoles 所在位置。
3. 接著，對這些亮的區塊以 Delaunay Triangulation 找出某個特定參數的 alpha-shape。完成之後會得到一組 contours(輪廓資料)，將這些

contours 進行 snake (active contour)演算法，以確認果蠅腦各層影像之 vacuoles 的位置與確切形狀。

4. 輸出各 contour 之資訊，並以人工方式刪除電腦誤判的 contours，再重新計算 vacuoles 的總體積。

根據上述研究步驟，我們在以下的敘述當中，決定本系統之應用需求：

1. 本系統為視窗式使用者介面。
2. 本系統專為此應用需求：**果蠅腦之老年痴呆情形分析**所設計，使得果蠅腦影像能夠依照上述之流程找出所有的 vacuoles 與體積計算之功能。
3. 本系統執行以後，使用者能夠依照視窗介面的提示，讀取一組由共軛焦顯微鏡取得的果蠅腦影像（特定格式之三維影像資料）。
4. 完成影像的讀取之後，使用者能夠便利地瀏覽各層影像資料。
5. 完成影像的讀取之後，使用者能夠依照視窗介面的提示，對所讀進的影像資料進行 Gaussian Matched Filter 演算法，在執行之前，使用者能夠設定 matched filter 的參數例如半徑，或是採用系統預設值。
6. 完成 Gaussian Matched Filter 之後，使用者能夠觀察處理過後的影像資料，並能切換至原先影像以比對處理前後之結果。
7. 完成 Gaussian Matched Filter 之後，方能讓使用者對於處理過後的影像進行 vacuoles 之搜尋動作。此部分之演算法包含 alpha-shape 搜尋與 snake，因此使用者在執行之前能夠設定 alpha-shape (最小半徑)與 snake (執行次數)之參數。
8. 使用者能夠將影像當中的 vacuoles (contours)資訊輸出至純文字格式檔案。
9. 由於上述演算法(Matched Filter 與 Contour Finding)可能會需要較長的執行時間（約數分鐘），故執行時系統須提供使用者目前執

行進度提示，並提供取消指令供使用者取消執行程序。

10. 完成所有的執行程序之後，使用者可從影像當中點選其中找出的 vacuoles，並將不滿意的 vacuoles 排除體積的計算。使用者亦能反悔，將排除的 vacuoles 復原並列入計算。
11. 第 10 項需求當中，每當排除／復原動作之後，系統應更新並提示使用者目前所有 vacuoles 之總體積。第 10、11 項需求之使用範例可參見圖 2-1 之(c)與(d)，紅色的輪廓為找到的 vacuoles。
12. 若對執行結果不滿意，使用者能夠清除所有的執行結果，並重新以不同的參數對影像資料執行所有功能。若清除原始影像，則會連同執行結果一併清除。

我們將於 3.2.1 節介紹本系統之**系統分析**，下圖為 Use Case Diagram

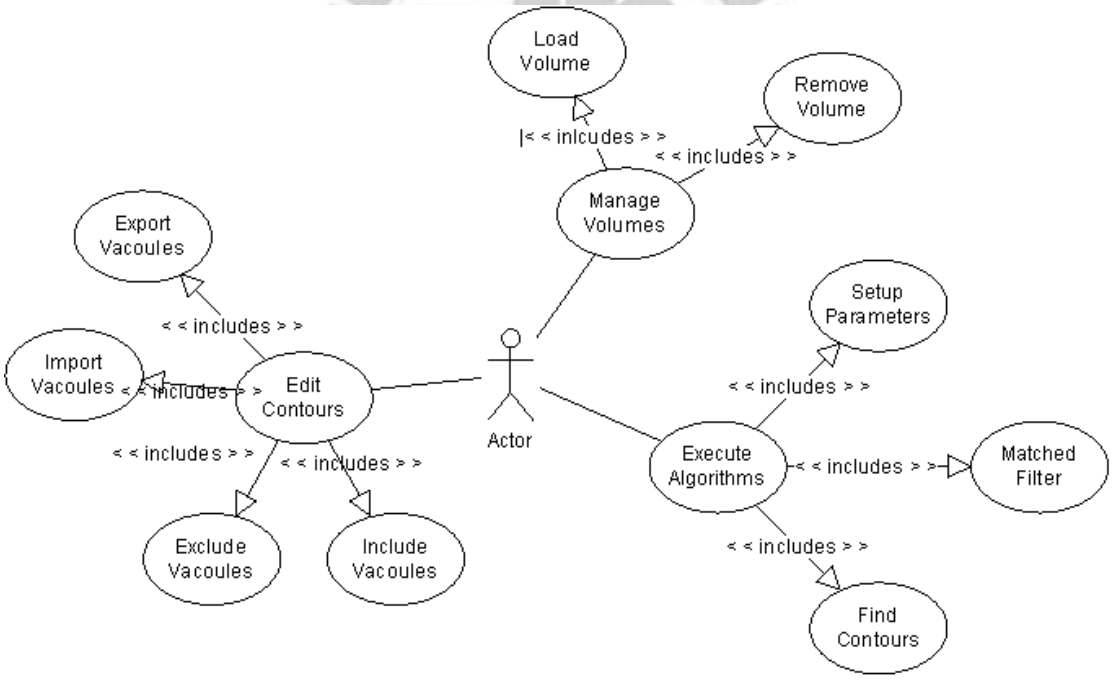


圖 3.1 Use Case Diagram of AlzheimerFly's UI

3.1.2 半自動果蠅腦神經追蹤工具－需求分析

我們由共軛焦顯微鏡取得果蠅腦神經影像，每隔某段間距取一張，形成三維的影像資料。果蠅腦神經影像以線狀為主，如圖 3.2 所顯示，影像為從頂端觀望所有切面之正交投影結果。

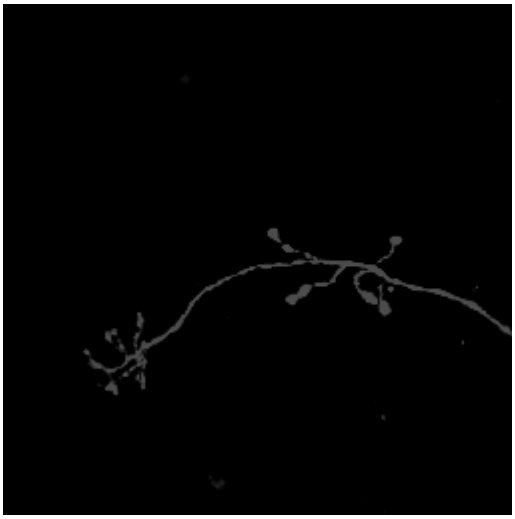


圖 3.2 果蠅腦神經組織影像範例

上圖為果蠅腦神經組織影像範例，從中可見神經組織以線狀結構為主，在此應用需求當中，需要盡可能將所有的神經線找出並標示出來，以便利神經結構重建、比對之應用。果蠅的大腦甚小，而共軛焦顯微鏡取像之垂直方向係機械微調方式移動取像位置，使得垂直方向之解析度受限。因此，儘管目前已有學者研究如何自動化的將這些神經線找出，在 z-sampling 解析度不足的情況之下，仍需要半自動的方法來完成此項工作。因此，我們需要設計一個便利的使用者介面程式，使得 Neuron Tracing 的工作能夠經由以下步驟完成：

Step 1: 輸入果蠅腦神經影像，為一組 Volume Data

Step 2: 計算此 Volume Data 之 3D-GVF，以[6]所提之演算法。

Step 3: 完成之後，以手動方式編輯神經線資料。

Step 4: 將編輯之後的神經線以 snake 演算法配合 3D-GVF 資料將神經線微調至完全附著於影像中的邊線上。

根據以上的研究步驟，我們在下段的敘述當中，決定該系統之應用需求：計算出 volume 的 gvf，並能以切換的方式觀看 gvf 和原始影像

1. 本系統為視窗式使用者介面。

2. 本系統專為此應用需求：**半自動果蠅腦神經追蹤**所設計，使得果蠅腦影像能夠依照上述之流程以半自動的方式標示出位於影像中的神經組織。
3. 本系統執行以後，使用者能夠依照視窗介面的提示，讀取一組由共軛焦顯微鏡取得的果蠅腦影像（特定格式之三維影像資料）。
4. 完成影像的讀取之後，使用者能夠便利地瀏覽各層影像資料。
5. 由於上述演算法(Matched Filter 與 Contour Finding)可能會需要較長的執行時間（約數分鐘），故執行時系統須提供使用者目前執行進度提示，並提供取消指令供使用者取消執行程序。
6. 完成 GVF 的計算之後，使用者可以以點選的方式標示出一個 neuron 的 initial path
7. 若已經有多個 neuron 呈現於螢幕，使用者可點選任意的 neuron，若與滑鼠座標最接近的 neuron (點到所有線段的距離的最小值)則被選取。
8. 若其中一個 neuron 已被選取，使用者能夠對其執行 snake 的演算法(可決定是否使用 gvf)
9. snake 可以儲存到檔案當中，可供事後讀取之用。
10. 讀取 snake 檔案是從已經開啟的 volume data 輸入；該 snake 的所屬的 volume 資訊(width/height/nSlices)應該與原 volume 相同。因此該 snake 的檔案當中應該具有所屬 volume 的資訊(屬性)。
11. 若對執行結果不滿意，使用者能夠清除所有的執行結果，並重新以不同的參數對影像資料執行所有功能。若清除原始影像，則會連同執行結果一併清除。

我們將於 3.2.2 節介紹本系統之**系統分析**，下圖為 Use Case Diagram

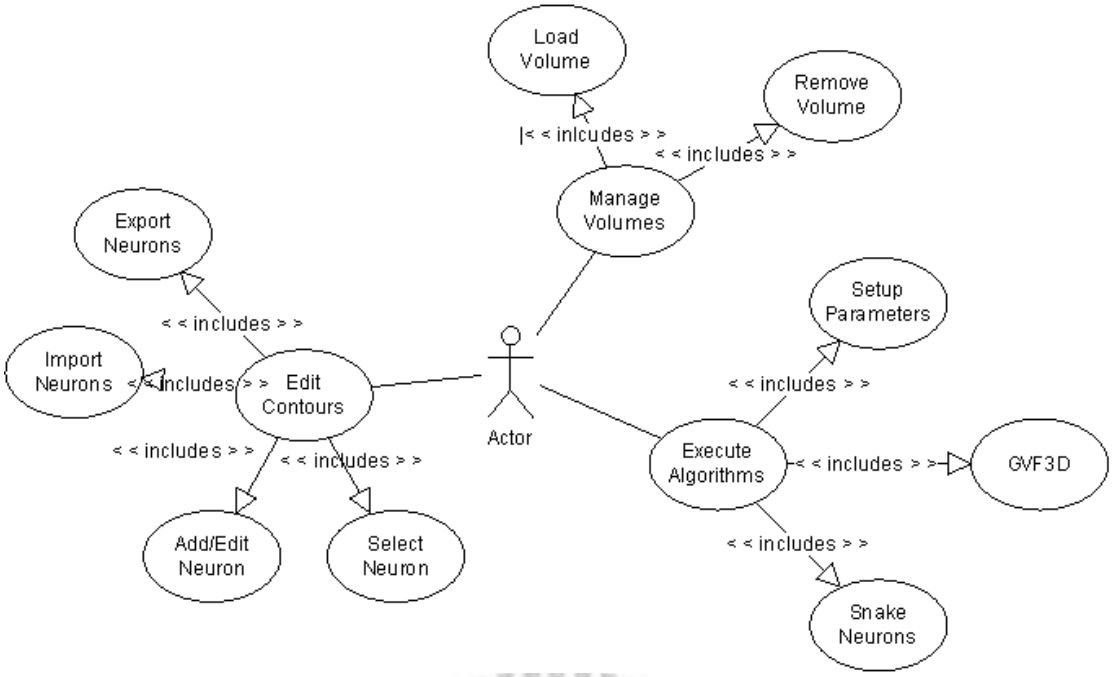


圖 3.3 Use Case Diagram of Neuron Tracing UI

3.1.3 使用者介面設計簡介

下圖為我們期望為本系統所設計的使用者介面：

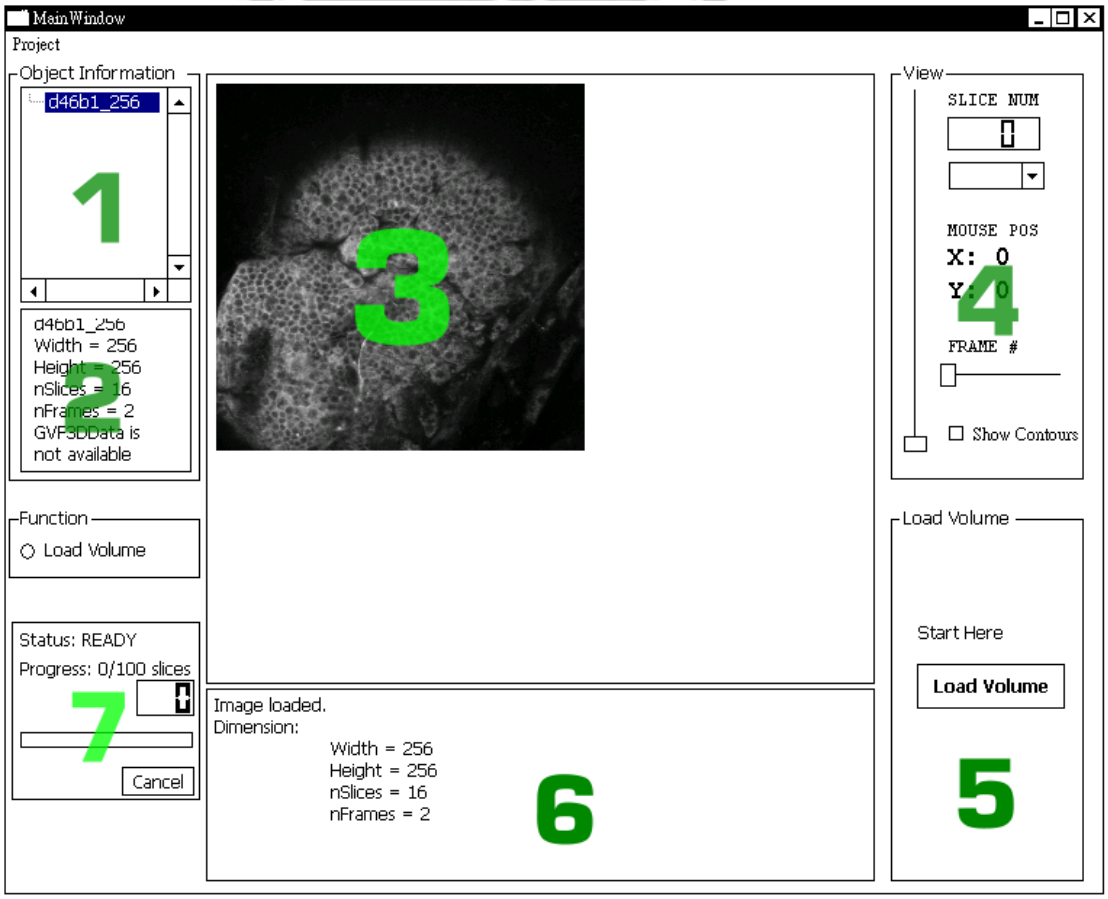


圖 3.4 User Interface Example

使用者以圖形化的方式，經由本元件－User Interface 操作系統的所有功能，包括影像讀取與儲存、影像瀏覽、影像處理、影像特徵資訊編輯瀏覽等。以 3.1.2 中的第 3、4 項需求為例（讀取一組 Volume Data 並能瀏覽各層影像），使用者從圖 3.4 當中的數字 5 所指示的視窗，以滑鼠左鍵按下”Load Volume”字樣之按鈕，即會彈出對話框供使用者選取影像檔案。選取之後，圖 3.4 的第 7 個視窗會顯示檔案讀取的進度。完成讀取的動作之後，視窗 6 將會顯示已經讀取完成之訊息，並出現該影像資料之相關資訊。之後數字 3 之視窗即會顯示其影像檔的第一層影像資料，使用者可從視窗 4 的控制器之圖像瀏覽各層的影像。

3.2 系統分析

在本節中，我們將介紹本系統的主要組成。其實，在這兩個應用需求的範例當中，大致上的結構其實是很相似的。我們在 3.2.1 節介紹兩系統的共同架構，3.2.2 節則是系統中各個組件之間之互動情形。

3.2.1 系統的基本組成

在本系統中，我們依據功能與屬性（使用者介面－應用需求－資料處理與作業系統）的不同，大致上分為以下數個元件，並分為四個層級，請見以下的 Package Diagram：

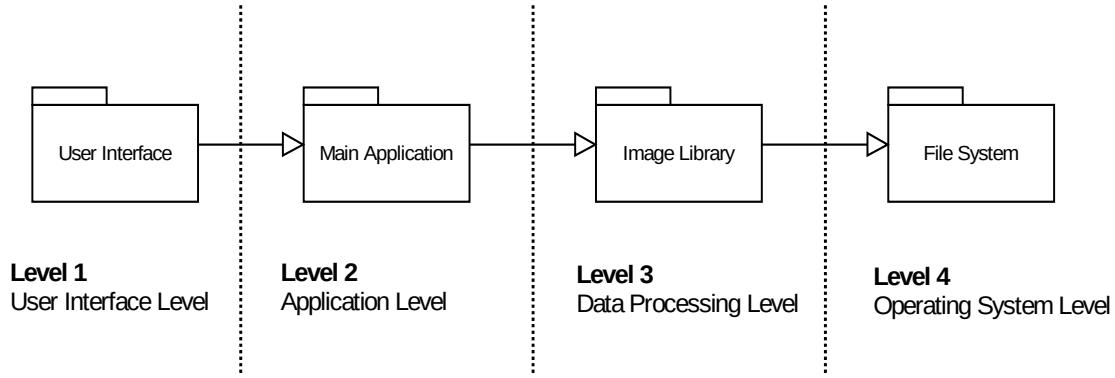


圖 3.5 Main Package Diagram

Level 1 至 Level 3 均為本系統的實作部份，分別為第一至第三階段的實作（見 1.3 與第 4 章）。本系統由上述的元件（Packages，主要共 4 個）組成，共分為使用者介面、應用需求、資料處理與作業系統等 4 個層級，其相依的關係如上圖所述，每一層的元件均會操作下一層的元件，而隔層之間則互不干擾。以下簡述各層之元件的主要工作：

一、使用者介面層級：

本層級的實作，主要為提供使用者一個圖形化的介面，讓使用者能夠以滑鼠或鍵盤操作本系統的各項功能。本層級的 User Interface Package 即用來接收來自使用者的各項指令，將其解讀成系統當中對應的功能之後，對第二層－應用需求層的元件執行相關的命令，並將執行結果或是資料呈現予使用者瀏覽。

二、應用需求層級：

本層級的實作，則以提供應用需求所需的各項功能為主。以本系統為例，在 3.1 當中提出了兩個應用需求，分別為**果蠅腦之老年痴呆情形分析**與**半自動果蠅腦神經追蹤系統**。本層級的唯一元件－Main Application 係針對某種應用需求（例如上述兩者之一），存放與「應用本身」相關的資料並明

確定義此應用相關的功能。Main Application 經由操作第三層－資料處理層的元件以達成該應用需求的所有工作。

以**半自動果蠅腦神經追蹤系統**為例，在本層級當中，資料的部分將包含一個或是數個 Volume Data，或是某個 Volume Data 之 GVF 的計算結果。而功能的部分則為讀取或是加入／刪除一組 Volume Data、切換至某一個 Volume Data、計算使用中的 Volume Data 之 GVF，或是在一個 Volume Data 加入／刪除／編輯一個 Neuron（神經線）等。

三、資料處理層級：

本層級的實作以影像資料存放與處理為主，定義二維與三維影像的資料結構，與存放現有的影像處理或是幾何計算相關的演算法，以提供應用需求層的 Main Application 元件操作之用。在本層的實作，則不與任何應用需求相關，僅提供適當的資料結構與一系列的演算法實作程式，並稱之 Image Library 元件。

其中 Image Library 係提供用來存放與操作 Image 與 Volume Data 的資料結構，並藉著 File System(檔案系統)提供影像讀取／儲存功能。Image Library 除了進行影像的輸入／存放／輸出以外，亦可執行影像的處理工作。

在 Image Library 當中，擁有許多影像處理或幾何計算之演算法的實作，從最基本的影像處理例如 Brightness/Contrast、Morphological Filters、Edge Detection、或自行定義的 Mask Convolution，與幾何計算的 Delaunay Triangulation、Convex Hull 與 Alpha Shape 或是與某些應用需求相關的實作例如 Matched Filter、GVF 或是 Snake 等。

四、作業系統層級：

本層級並不屬於本系統的實作部份，而是作業系統本身（例如 Windows XP 或是 Linux）。其中 File System 則是該 OS 所提供的檔案系統。本系統的

Image Library 則是藉由操作檔案系統以實現影像的資料讀取與儲存。

3.2.2 系統各組件的互動模式

使用者指令與各層級之間的互動機制如下圖：

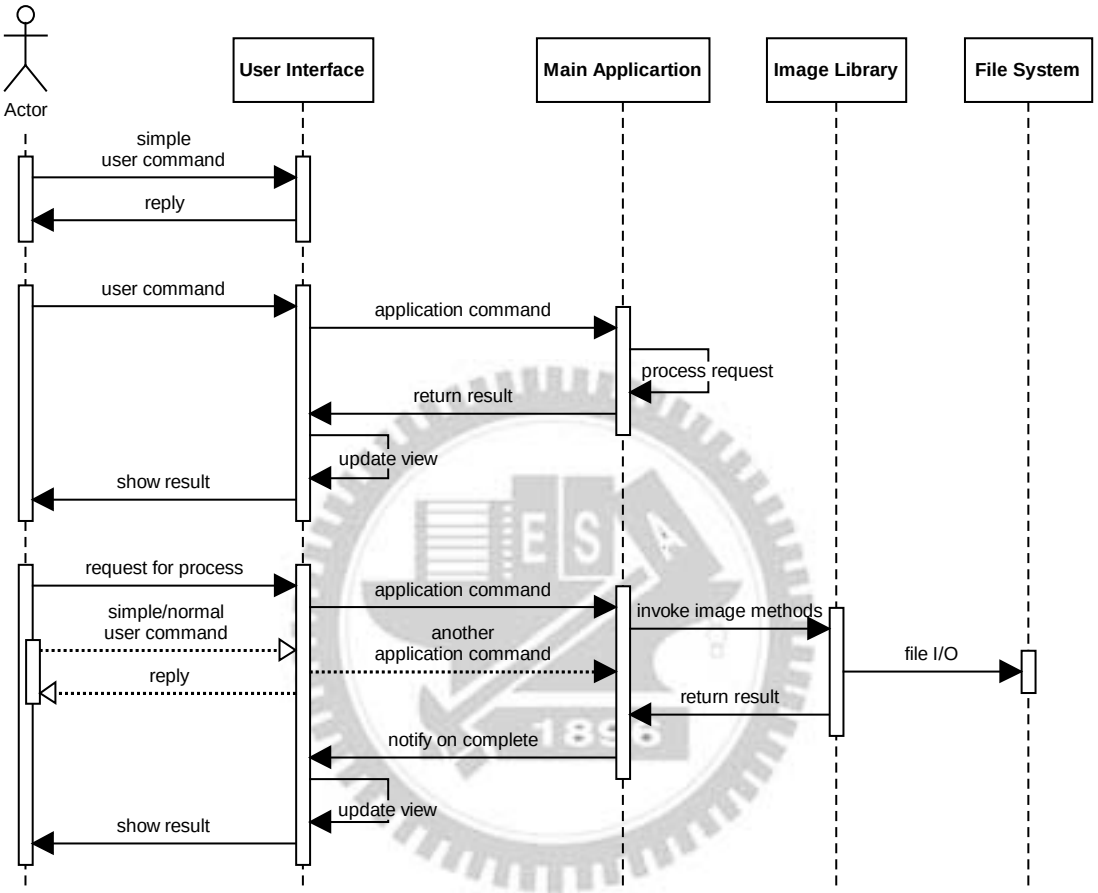


圖 3.6 Top Level Sequence Diagram

來自使用者的指令，從簡單到複雜，大致上可分為三種：

1. 簡單的指令 Simple User Command，例如滑鼠移到顯示影像之視窗的某處，系統則在 View Panel 上面更新顯示的座標。此時直接在 UI 的程式當中即可處理完成，並不需要接觸到應用需求層級的部分。
2. 一般的使用者命令 User Command，例如在 View Panel 的控制棒移動瀏覽的影像之索引(第幾張 slice)，UI 程式接收到了移動控制棒的訊息，轉換至應用需求當中的對應命令（例如更換目前瀏覽的索引值和更新影像

的內容）至 Main Application。完成命令之後則將影像視窗的顯示更新。

3. 長時間行程的使用者命令 Request for Process，通常這個指令以需要較長的處理時間（超過 5 秒或長達數分鐘），例如 Matched Filter、Vacuoles Finding[1]或是 GVF 的計算。從圖 3.6 當中，使用者的 Request for Process 即是對於本系統提出一個處理程序的需求，提出此需求，一直到最後來自 UI 的 show result 即是完成該處理程序。

當 request for process 發生之後，UI 對 Main Application 作出對應的 application command(與應用需求相關的命令，例如計算 Volume Data 的 GVF)，接著 Main Application 找出目前使用中的（即使用者正在瀏覽觀看的）Volume Data，並對 Image Library 發布處理該組影像的訊息與對應的功能之後，藉由 Image Library 將此 Volume Data 轉換至相對應之演算法實作程式的輸入資料(input data)並始執行演算法。完成後，將結果傳回 Main Application。

從 request for process 一直到 show result 可能經歷數十秒至數分鐘之久，因此在 actor 即 user 與 UI 之間，亦插入了一個 simple/normal user command，並以虛線表示互動的關係。這表示使用者在等待處理程序的時間，亦可對於本系統執行上述 1 與 2 類的指令（例如點選顯示影像當中的 neuron 或是瀏覽影像等指令），不需要等待至處理程序完成，即可使 UI 立刻對使用者作出回應。

以上以概觀的角度介紹本系統的各個元件組成，在下一章中，我們將針對各個元件，提供細部設計的介紹。

第四章 系統設計

在 3.2 節，我們提到本系統中，從使用者介面、應用需求、資料處理至作業系統等層級等各個元件之組成與互動機制。除作業系統層級並非本系統實作範圍之外，在本章中，我們將針對本系統中各個元件提供細部設計的介紹。本章分為 4 個小節：4.1 節介紹資料處理層的 Image Library 之設計；4.2 節則分別為**果蠅腦之老年痴呆情形分析、與半自動果蠅腦神經追蹤**兩個不同的應用需求之 Main Application 元件個別介紹；另外 User Interface 元件將於第五章連同系統展示介紹其設計。

4.1 Image Library

4.1.1 基本功能需求

本節將介紹用來操作影像資料，包含讀取／儲存以及執行影像處理演算法的 Image Library 之設計部份。此元件 Image Library 期望能夠達成以下的功能：

1. 存放二維的 Image 或是三維的 Volume 之影像資料。
2. 提供影像資料的取得、更新(read/write access)，或是瀏覽影像中的每個像素(類似 STL 當中的 iterator 概念)
3. 將影像資料執行現有的影像處理演算法(即 image processing)。
4. 從檔案系統當中讀取或儲存現有的影像(可為 bmp, jpg 或是儲存 Volume Data 之 lsm 格式等等)
5. 產生一組新的空白的影像（使用動態記憶體配置），並決定尺寸或是像素規格（例如長寬高，是否為 Image 或是 Volume，一個像素多少 byte 組成）。
6. 複製影像資料至新的一組影像。

4.1.2 Image Library 影像資料結構

在 Image Library 當中，影像的格式分為兩種，分別為二維的 Image 與三維的 Volume Data，並以未經壓縮的點陣圖 (uncompressed bitmap) 格式存放。影像由像素組成，一個像素表示一個顯示於任何顯示裝置的一個點(dot)。我們先介紹在 Image Library 當中，兩種影像資料的表示方式：

a. 二維影像表示法

二維影像稱作 Image 由像素組成。影像中的所有像素以矩陣的方式排列而成，因此具備了最基本的兩項屬性：寬與高(Width and Height)。令寬為 w，高為 h 的影像，則具有 w * h 個像素(pixel)。一個像素由 1 至 4 個 byte 表示 (8 至 32 bit)，數值越大表示亮度越高。因此該張影像所佔的位元組個數為 w * h * 一個像素所佔的位元組。表示法的部分請見下圖：

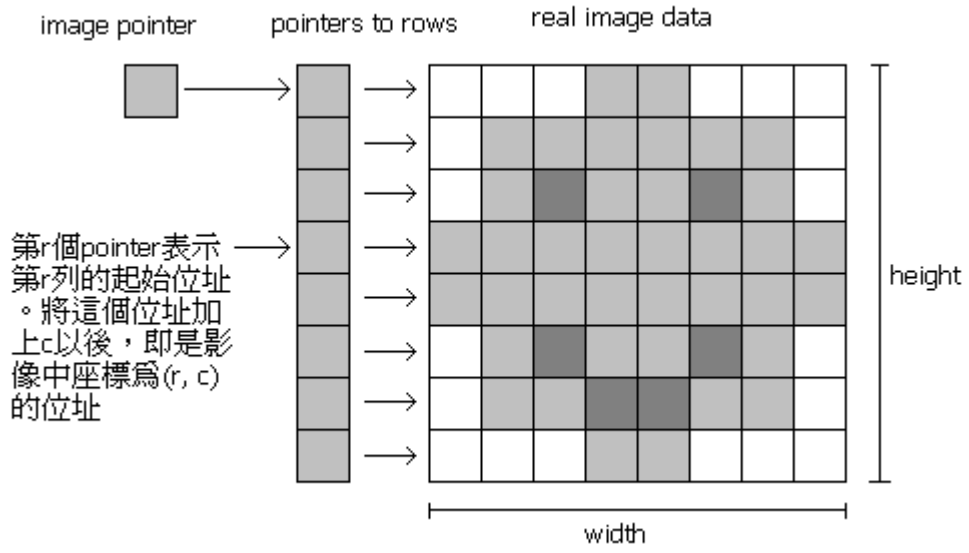


圖 4.1 Image 資料結構表示法

(a) image pointer (b) pointers to rows (c) real image data

假設目前有一張 width * height 的影像，正如圖 4.1 所述，width = height = 8，每個像素為 8-bit (因此亮度可表示的範圍為 0 至 255)。則此張影

像共 8 * 8 = 64 個像素，佔用 64 個位元組。在 Image Library 當中，該影像以大小為 8*8 = 64 個元素的一維陣列 (在此每個像素的型態為 unsigned char)儲存。

除此之外，為了便利影像資料的定址(addressing)，在此影像範例中，有一組大小為 8 (即 height)的一維陣列，其中每個元素存放 image 當中各個 row(列)的起始位址，稱為 pointers to rows。亦即，pointers to rows 當中第 k 個元素表示影像中第 k 列的起始位址。而 pointers to rows 此陣列的起始位址的指標記為 image pointer。因此，存取該影像第 r 列的第 c 個像素，可以 imagePointer[r][c]表示，如圖 4.1。

補充說明：圖 4.1(c)的影像排成方形，實際上是 64 個元素的一維陣列。另外，imagePointer 是一個指標變數，記錄 pointers to rows 的起始位址；imagePointer[k]表示 pointers to rows 的第 k 個元素，即影像中第 k 列的起始位址。另外，二維影像資料結構的記憶體配置法如下，以 C++語法寫成：

```
unsigned char* imageData = new unsigned char[height * width * bytesPerPixel];
unsigned char** imagePointers = new unsigned char* [height];
for (int row = 0; row < height; row++, imageData += width * bytesPerPixel)
      imagePointers[row] = imageData;
```

Code Listing 4.1 Allocating an image object

b. 三維影像表示法

一組三維影像(volume data)，由多張二維影像堆疊而成。因此，除了 Image 既有的寬(width)與高(height)以外，另外又多了切面數(在此記為 depth)屬性。一組切面為 d，寬與高分別為 w 與 h 的三維影像，佔用記憶體的空間則為 d * w * h * 一個像素的位元組數(bytesPerVoxel)。三維影像的像素記為 voxel，而二維的影像像素則為 pixel。表示法的部分請見下圖：

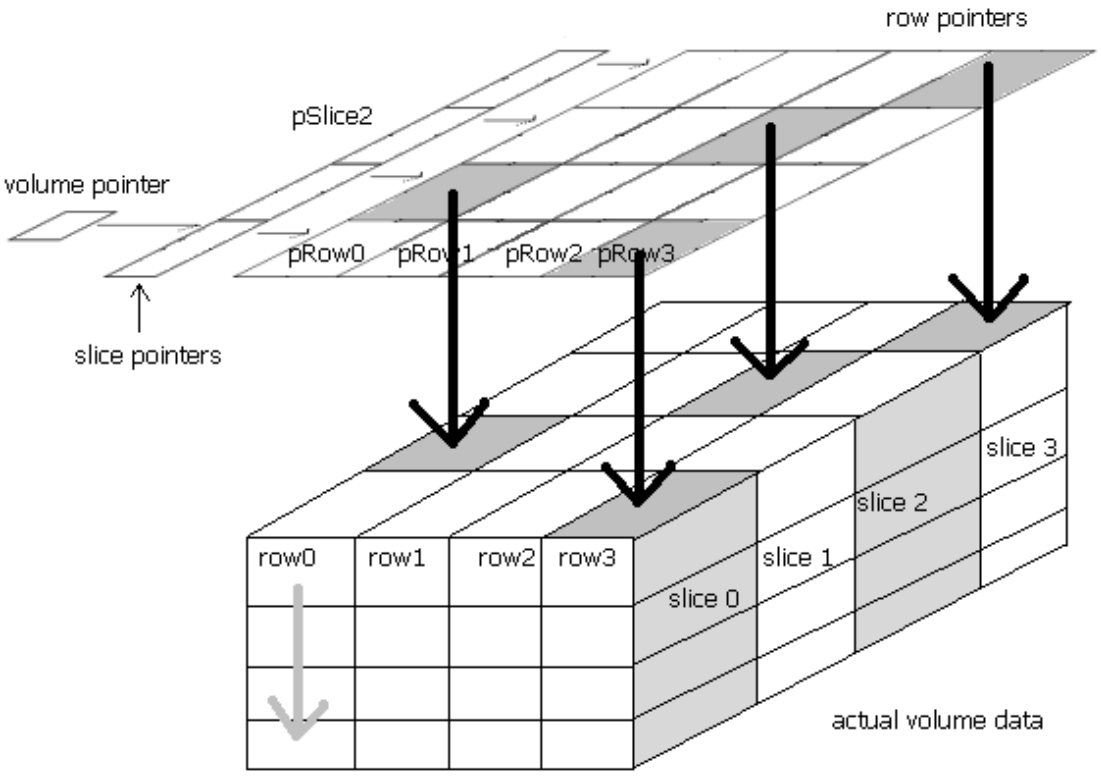


圖 4.2 Volume 資料結構表示法

圖 4.2 的上半部與圖 4.1 相似，只是在二維影像存放像素的陣列當中每個元素由 unsigned char 型態以 unsigned char* (即指標)取代，成為這個 volume 各個 row(列)的起始位址。因此，對於一組由 d 張寬 w 高 h 的影像所堆疊而成的 volume data 而言，共有 d * h 個 rows(列)與 s 張 slices。

在上圖中，此 volume data 將會有 d * h 個 row pointers 與 s 個 slice pointers。最靠近 volume data 的一維陣列，是 d * h 個 row pointers，頭 h 個 row pointers 屬於第一張 slice (slice 0)的 h 個 row 的起始位址，第 h + 1 至第 2h 個 row pointers 則屬於第二張 slice，以此類推。

而每一張 slice 所屬的 h 個 row pointers，均有一個 slice pointer 指向該 row pointers 的起始位址，因此 slice pointer 應該有 d 個，稱為 slice pointers。而這些 slice pointers 的起始位址則由 volume pointer 存放。

影像資料的存取方式

若要存取在上圖中第三張 slice (即 slice 2，編號均由 0 開始計算)，當中的位於第三條 row (即編號 2)的第 1 個像素 (即編號 0)，則

1. 首先查閱 volume pointer 取得 slice pointers 的起始位址
2. 接著從 volumePointer[2] (即是 slice pointers 的第 3 個元素) 取得該 slice 的 row pointers 之起始位址(此時比照二維影像)
3. 接下來從 volumePointer[2][2](即該 slice 的 row pointers 第 3 個元素) 取得第三條 row 的起始位址。
4. 最後以 volumePointer[2][2][0]取得 volume 當中 slice 2 位於 row 2 的 pixel 0 的數值。

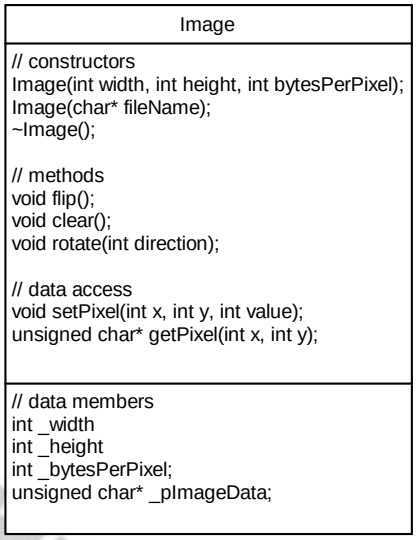
多組 Volume Data 的影像資料

若影像資料為多組的 volume data，例如一組果蠅腦細胞(以共軛焦顯微鏡拍攝的一組 volume data)隨著時間的變化情形，可能需要以多組的 volume data 記錄，因此這種 volume data 除了 depth 以外，又多了畫面數(nFrames 即 number of frames)屬性。若有一組畫面數為 f，切面數為 d 的 volume data，則有 f * d 張 slices，因此 slice pointers 共有 f * d 個元素；而在此 volume data 中，亦有 f 個 frame pointers 個別指向各 frames 的 slice pointers 之起始位址。因此，若要存取 frame **i**, slice **j**, row **r**, column **c** 的像素內容，則需以 volumePointer[i][j][r][c]存取之。

以上介紹二維與三維影像資料的表示法。在 4.1.2 中，我們將介紹 Image Library 當中，Image/Volume Data 之設計方式。

4.1.3 來源(Source)／內容(Content)／接觸介面(Access)設計模式

請參考以下的設計：



這是一個設計用來操作影像資料的一個類別(class)，稱作 Image，在第一個 constructor 當中，該 image 可經由指定 width 與 height 和 bytesPerPixel，以動態記憶體配置方式完成影像物件的建立，或是從讀取某種特定格式的影像檔案來建立影像物件(第二個 constructor)。

除此之外，該影像可透過 get/setPixel 來實現影像資料的存取，並可經由 flip、clear 與 rotate 等 method 來操作影像的翻轉、清除與旋轉等動作。我們明顯可以看出，這個類別已定義了該影像物件的屬性、行為(操作)與建構方式。然而，若需要延伸該物件的建構方式（例如來自其他檔案格式）、操作（例如加入一些影像處理的演算法實作）或是資料存取方式（例如存取 subpixel），則維護該物件的實作將會較為困難。畢竟影像物件的用途甚廣，就算已經擁有了相當完整的建構、存取與操作等實作，仍可能存在未知的需求。

因此在本系統的 Image Library 當中，我們將影像的建構、資料儲存、資料存取與操作分為三個獨立的類別(class)。Image Library 的設計以下圖的概念為基礎，實作出一組用來處理二維與三維影像資料的**類別函式庫**(class library)以達成 4.1.1 所述之功能，並期望在未來出現新的應用需求時，能夠容易地維護與延伸物件的實作。

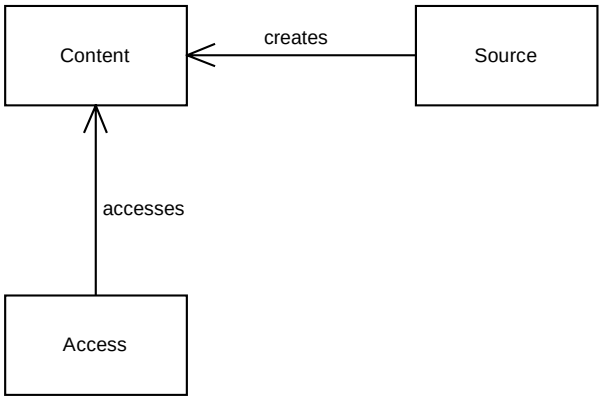


圖 4.3 Source/Content/Access Pattern

上圖表示 Image Library 用來儲存、操作、建構影像資料的主要物件，分為三大類： Content（內容）、Source（來源）與 Access（資料操作）。每個類別均分為 Image 與 Volume，即 Content 分為 ImageContent 與 VolumeContent，其餘以此類推。Image 與 Volume 分別處理二維與三維影像資料，兩者獨立。此三類物件的功能依據性質有所不同，以下我們將個別介紹之：

a. Image/Volume Content: 影像內容。專門用來儲存影像資料與相關屬性，例如各個維度之尺寸與各個 row、slices 與 frames(後兩者用在 volume)的排列方式，並提供查閱屬性資訊的方法(methods)與影像資料的起始位址(image bits)。設計如下：

ImageContent

int getWidth();
int getHeight();
short getBytesPerPixel();
int getWidthBytes();
bool rowsUpsideDown();
unsigned char* getImageData();

ImageContent(ImageSource& imageSource); ImageContent(ImageContent& imageContent); ~ImageContent();

VolumeContent
int getWidth() int getHeight() **int getDepth()** **int getnFrames()** short getBytesPerVoxel() int getWidthBytes() bool rowsUpsideDown() **bool slicesUpsideDown()** **bool framesUpsideDown()** unsigned char* getVolumeData()
VolumeContent(VolumeSource& volumeSource) VolumeContent(VolumeContent& volumeContent)

圖 4.4 Class ImageContent and Class VolumeContent

Image/Volume Content 本身提供記錄影像資料與相關屬性資訊的功能。就 ImageContent 而言，屬性除了 width/height/bytesPerPixel 以外，另外提供 widthBytes 表示每條 row 所佔用的位元組數以便利計算之用，以及 rowsUpsideDown 之 true/false 值，表示影像的各 row 之排列方式，在記憶體中是從 row 0 排列至 row height - 1 亦或順序顛倒排列(例如 Microsoft®的 bitmap 影像格式)。而 VolumeContent 則多了圖中粗體字部分，即 depth, nFrames 與 slices/framesUpsideDown，也表示各 frame 或是 slices 的排列順序。

Image/Volume Content 的建構方式相當單純。其中一種是傳入一個繼承於 Image/Volume Source 的物件，而影像物件的建構方式則定義在傳入的 Source 物件本身。

另外一種方式則為傳入一個 Image/Volume Content 的物件，表示從來源的 Image/Volume Content 複製一個相同的影像物件。

b. Image/Volume Source: 影像來源，專門用來建立影像資料(Image/Volume Content)的物件。在 Image Library 當中，所有和影像建立有關的物件，都繼承於此類別。舉例而言，假設我們自行設計一個用來讀取 bmp (Microsoft® Bitmap) 圖檔的物件稱為 MyBMPReader，並繼承於 ImageSource，那麼，若要藉由此物件讀取一個 bitmap 格式的影像檔案至一個 ImageContent 當中，則以下的程式片段即可達成。

```
01 MyBMPReader reader(“image.bmp”);
02 ImageContent image (reader);
```

Code Listing 4.2

Constructing an Image Object from an Image File

程式片段中的 01 行表示產生一個 MyBMPReader 的物件稱為 reader 作為**影像來源**，並且指定 image.bmp 使其能夠將 image.bmp 影像檔案傳輸至指定的 ImageContent 當中。讀取的動作在 02 行完成。02 行能夠解讀成藉由來源物件 reader 產生一個叫做 image 的影像物件。這個 reader 物件的實作將會讀取

image.bmp 檔案，並將影像內容與相關屬性轉換至 image 物件當中。

我們在此介紹影像建構時 Source 與 Content 的互動機制。

執行 Code Listing 4.2 的兩行程式時，在 01 行當中建立一個影像來源並址定相關參數（例如檔案名稱），真正的建構程序，則是在 02 行：

```
02    ImageContent image (reader);
```

ImageContent 除了擁有影像資料的起始位址以外，亦包含影像的屬性資訊。因此 ImageContent 在建構(執行 constructor)的程序當中，需要從傳進的 ImageSource 物件取得即將建構的影像物件之所有資訊。這些資訊即是圖 4.4 下方的資料成員，為便利起見，我們將這些資料成員封裝成 ImageCreateStruct 結構，讓 ImageSource 完成影像(從檔案或是 OS)的建立以後回傳此結構，再指定到 ImageContent 的資料成員中。下圖介紹影像建立的過程，從 ImageContent 的建構子(constructor)呼叫 ImageSource 的建構程序直到取得影像資訊為止：

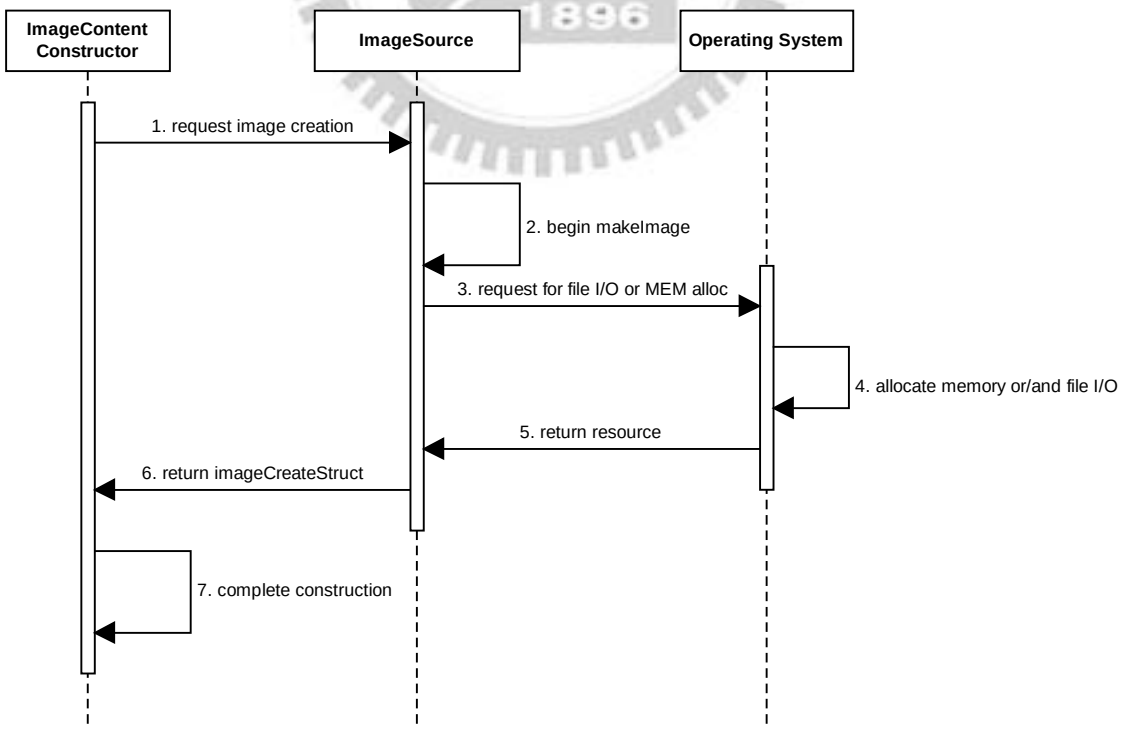
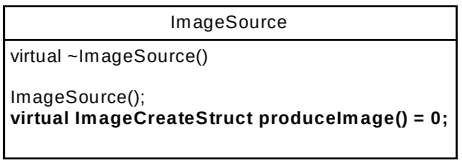


圖 4.5 Sequence Diagram of Image Creation

在 ImageContent 的 constructor 當中，起初對於傳入的 ImageSource 作出建構影像的請求(呼叫 produceImage 函式)，ImageSource 於是依據原先指定的參數（例如檔案名稱）開始以本身特定的方式(例如，程式片段 4.2 當中的 MyBMPReader 將從指定的影像檔案讀取資料)產生影像資料與相關屬性，完成後回傳 ImageCreateStruct 至 ImageContent 的建構子，最後將影像資料與屬性指定至物件當中完成建構。

下圖為 ImageSource 的設計：



影像產生的方式，由虛擬函式 produceImage 當作取得影像資訊的介面。因此任何繼承於 ImageSource 的物件，必須提供圖中 produceImage 函式的實作。如此進行圖 4.5 的第 2 個 step 時，ImageSource 物件會呼叫自己的 produceImage，但實作部份卻在繼承於此之物件，例如定義在 MyBMPReader 的 produceImage 函式中。

另外，其中 VolumeSource 的設計與 ImageSource 相同，只是將 Image 改成 Volume 即可。

c. Image/Volume Access: 影像資料存取，專門用來便利影像資料取得與操作的物件。ImageContent 當中，僅提供影像資料的起始位址，這樣的設計在於存取方式可以自行定義，較為不受限制。在 Image Library 當中，我們則提供 Image/Volume Access 定義預設的影像資料操作方式。如 4.1.2 影像資料表示法所述，Image 的 row pointers、Volume 的 slice 與 frame pointers 均定義於 Image/Volume Access 當中。請見以下本物件的設計：

Image Access<PIXEL_TYPE>
ImageAccess(ImageContent& imageContent); ~ImageAccess(); int getWidth() int getHeight() int getDepth() int getnFrames() short getBytesPerVoxel() PIXEL_TYPE* getImageData()
operator ImageContent&() PIXEL_TYPE* operator [] (int i)

Volume Access<VOXEL_TYPE>
VolumeAccess(VolumeContent& volumeContent); ~VolumeAccess(); int getWidth() int getHeight() int getDepth() int getnFrames() short getBytesPerVoxel() VOXEL_TYPE* getVolumeData()
operator VolumeContent&() VOXEL_TYPE*** operator [] (int i)

Image/Volume Access 以 template(樣板)型式提供使用。Access 物件均包含 Image/Volume Content 的參考(reference)，除了可直接參考 Image Content 的屬性與影像資料以外，另外提供了一個 operator []，能夠以多維陣列的方式直接對影像作操作。若要以 Access 操作一張存在的 ImageContent，則以下的實作即可使用。

```
01  MyBMPReader reader(“image.bmp”);
02  ImageContent image(reader);
03  ImageAccess<unsigned char> myAccess(image);
04  myAccess[10][7] = 15;
05  unsigned char* p = myAccess[7];
```

Code Listing 4.3 Image/Volume Access

範例中 03 行表示產生一個 Access 物件，用來存取影像物件 image，並且指定像素(pixel)的型態為 unsigned char(在 C 語言中，通常為佔用一個 byte 的變數)；04 行則指定影像中的 row 10 之像素 7 的數值；05 行則取得 row 7 的起始位址至指標變數 p。下列為 Volume Data 的使用範例

```
01 MyLSMReader reader(“volume.lsm”);
02 VolumeContent volume(reader);
03 VolumeAccess<unsigned char> myAccess(image);
04 myAccess[0][7][1][2] = 15;
05 unsigned char* pRow = myAccess[0][2][4];
06 unsigned char** pSlice = myAccess[0][1];
07 unsigned char*** pFrame = myAccess[1];
```

04 行指定位於 frame 0, slice 7, row 1, column 2 的像素亮度值。

05 行取得 frame 0, slice 2, 的 row 4 那列像素之起始位址(即 row pointer)

06 行取得 frame 0, slice 1 之存放 row pointers 之起始位址(即 slice pointer)

07 行取得 frame 0 之存放 slice pointers 之起始位址(即 frame pointer)

以上介紹 Image Library 當中的影像物件表示法以及影像物件之設計。在 4.2 當中，我們將以本專案的兩個應用需求，各別設計屬於其之 Main Application 元件，以供 User Interface 操作之用。

4.2 Main Application 之應用需求層級元件設計

在本節中，我們將介紹 Main Application 元件的設計。在本專案中，有兩個應用需求，分別為**果蠅腦之老年痴呆情形分析**與**半自動果蠅腦神經追蹤系統**。在本元件中，主要的物件除了存放目前應用程式當中所有的 Volume Data 物件，與電腦程式計算的結果(例如果蠅腦的空洞區域 Vacuoles 資訊)或是使用者編輯的資料(例如已編輯的 Neuron 資訊)。

另外還提供了屬於該應用需求的所有功能之實作。例如在**果蠅腦之老年痴呆情形分析**中，藉由操作 Image Library，提供了 Matched Filter 演算法，以及 Vacuoles 搜尋程序；而**半自動果蠅腦神經追蹤系統**則提供計算 VolumeData 之 GVF，與加入／刪除／編輯某個 VolumeData 之 Neuron Path。在本節中，4.2.1 介紹應用需求之共同部分之架構，4.2.2 與 4.2.3 則分別介紹兩應用需求的主程式之設計。

4.2.1 Main Application

在 Main Application 當中，對於任何應用需求，我們將本系統所有存放的 Volume Data 設置於此。我們提供一個容器(contatiner)的資料結構以存放之，是為一樹狀結構。Volume Node 容器的設計如下：

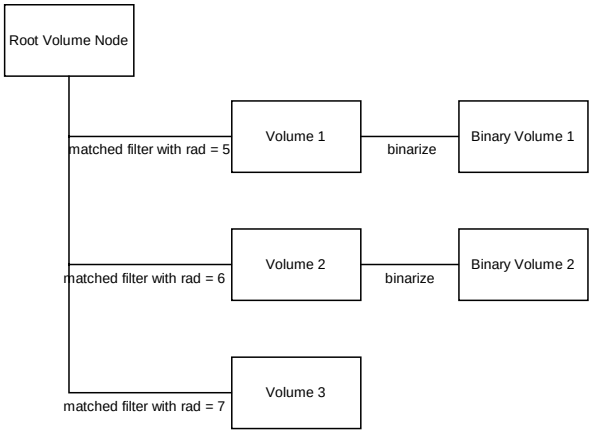


圖 4.6 Example Tree Structure of Volume Node

樹狀結構的根節點是本系統目前所要操作的主要 Volume Data，通常從檔案系統當中讀取現有的共軛焦顯微鏡影像資料，例如果蠅腦。而子節點表示針對此 Volume Data 作了處理以後的結果（例如 Matched Filter）之 Volume Data，之後亦可對這些 Volume Data 從事計算，也會延伸出新的 Volume Data 則記錄在該 VolumeData 的子節點（例如 BinaryVolume1 與 2，是將以 Matched Filter 計算出的 Volume Node 作 Binary Threshold 處理以後的 Volume Data）。以下介紹 Main Application 各 class 之設計：

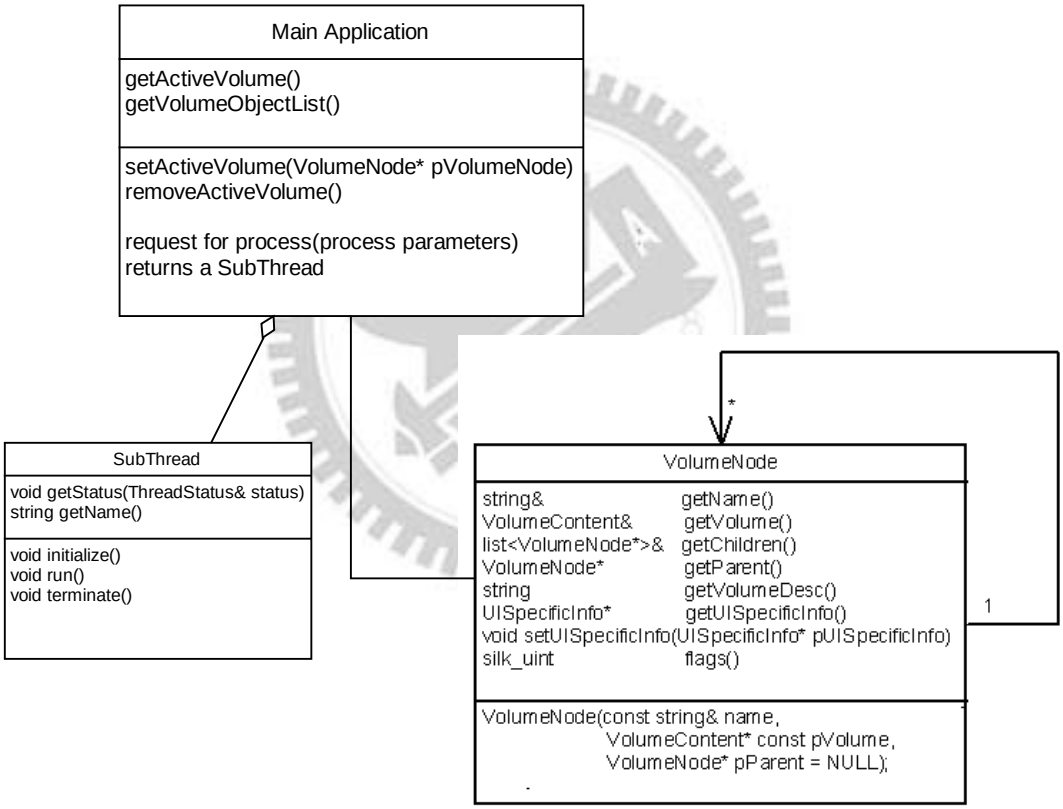


圖 4.7 Main Application Class Diagrams

以下介紹圖 4.7 當中所述的兩個 class：

Class VolumeNode

我們將樹狀結構的 Volume Data 容器存放在 Main Application 當中，因此 Main Application 將會包含一個 Root Volume Node，從該 Node 的 reference

能夠找出所有系統中的 Volume Data。對一個 Volume Node 而言，除了封裝 Volume Data 本身以外，也具備 Name(名稱)、Children(子節點參考)、Parent(父節點參考)、Description(說明)、flags(記號，例如表示其中包含的 Volume Data 是否與別的 Volume Node 共享)、以及 UISpecificInfo(與使用者介面相關的資訊)等屬性。

其中比較特別的是 UISpecificInfo，表示與使用者介面相關的資訊，均放在本屬性中。以本系統為例，其屬性資訊由 User Interface 元件當中定義，其中包含了該 VolumeNode 目前使用者瀏覽的影像切面位置(index of slices)與是否顯示該 Volume Data 之影像以外的資訊（例如在 AlzheimerFly 中的 Contours 以及在 NeuronTracing 的 Neurons）。

Class MainApplication 與作用中的 Volume Node (Active Volume Node)

在MainApplication當中，除了存放系統所有Volume Data以外，亦包含了作用中的Volume Node。就像使用MS-DOS©作業系統一般，一個磁碟機以目錄的方式作為檔案系統，而使用者在操作DOS之時，輸入的任何命令均是作用於當下的目錄(例如dir則是列出作用中目錄中所有檔案)。同樣的，在系統以樹狀連結成的Volume Node當中，也能夠從User Interface來選取與操作目前作用中的Volume Node。以本系統為例，User Interface在 3.1.3 數字 1 指示之視窗提供這樣的介面，在UI中接收使用者點選的Volume Data物件，經由操作MainApplication當中的setActiveVolumeNode函式改變目前作用中的Volume Node，或是執行removeActiveVolumeNode以刪除目前作用中的Volume Node。在不同的應用需求當中，也有可能藉由提出處理程序的需求（例如Matched Filter或是計算GVF），而對於作用中的Volume Node加以操作，並在該VolumeNode的Children當中加入計算的結果。

Class SubThread

SubThread 在本系統中，是 Command Pattern[9][10]的一個設計實例，稱為**子線程**。顧名思義，即是一個應用程式當中的某個處理程序。在本系統中，我們將與 Volume Data 相關的所有功能（例如讀取 Volume Data，或是計算 GVF）均定義於繼承此類別的物件當中。因此，也是透過繼承於 SubThread 的所有物件來操作 Image Library 元件當中影像處理部份的功能。User Interface 可以從 Main Application 發出執行程序的需求，並以線程(Threading，即主程式啟動一個 Thread，並執行之)的方式執行該程序，其步驟如下：

1. 取得一個執行程序的物件(SubThread*)
2. 填入該執行物件的參數(以 Load Volume 為例，填入的參數應為影像的檔案名稱)。並執行該物件的 establishParams 以初始化參數。
3. 執行 SubThread 的 initialize()函式，以初始化本程序。若未成功執行，則會回傳一個 C++ Exception (string 型式)
4. 執行 SubThread 的 run()，也就是正式執行該程序。
5. 中途如欲中斷程序，則執行 terminate()，但不會馬上停止，而執行中的程序會進行收尾的動作至完成。此部分為 Two-Phase Thread Termination Pattern[10]的設計實例。
6. 若 run()執行完成，則執行 complete()以進行完成程序之後的動作。例如將計算的結果加入作用中的 Volume Node 作為子節點。

此外，SubThread 在執行的過程中，可用 getStatus 方法得知目前程序的執行狀態（可將此呈現於 UI 上，以供使用者了解目前執行狀態）。而 SubThread 呈現的狀態有以下幾種：

NOT READY:　等待輸入參數。

INVALID:　無效的執行程序，可能是無效的參數或無法初始化所致。

VALID: 已指定參數，但尚未初始化。

INITIALIZING: 正在進行初始化

READY: 已完成初始化，隨時可執行。

RUNNING: 程序執行中，等待完成。

TERMINATING: 程序已被提出終止要求，進行收尾的動作。

TERMINATED: 程序已被終止。

ALMOSTDONE: 程序執行完成，但尚未進行最後工作。

COMPLETING: 程序正在進行最後的工作。

COMPLETED: 程序已完整的執行完成。

以下為本系統執行程序之狀態轉換(State Diagram)圖：

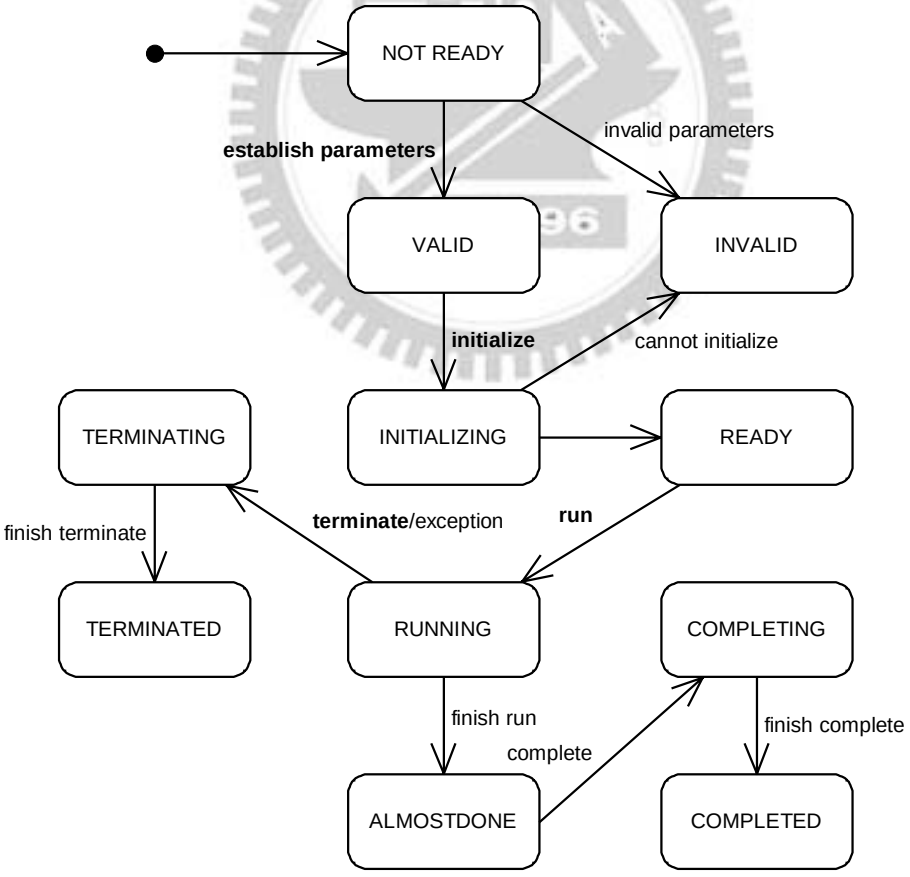


圖 4.8 State Transition Diagram of SubThread

上圖描述 SubThread 整個執行過程的狀態轉換情形。粗體字的事件表示來自外部(存取 SubThread 者)函式呼叫，而其他的事件則為 SubThread 自身函式引發的事件。剛取得 SubThread 物件時，呈現 NOT READY 的狀態。經由指定參數之後，呈現 VALID，之後進行初始化進入 INITIALIZING，若無法進行，則成為 INVALID。呈現 READY 狀態以後即可執行 run 而進入 RUNNING，若被外界終止則進入 TERMINATING，若已完成程序則進入 ALMOSTDONE 進而進入 COMPLETING 以完成程序。

以上介紹 Main Application 的主要物件，接下來的兩小節將依據兩應用需求個別介紹屬於各自的設計。

4.2.2 應用需求：果蠅腦之老年痴呆情形分析－AlzheimerFly

對此應用需求而設計的 class 命名為 AlzheimerFlyApplication，並繼承於 MainApplication。下圖為 AlzheimerFlyApplication 的 class diagram:

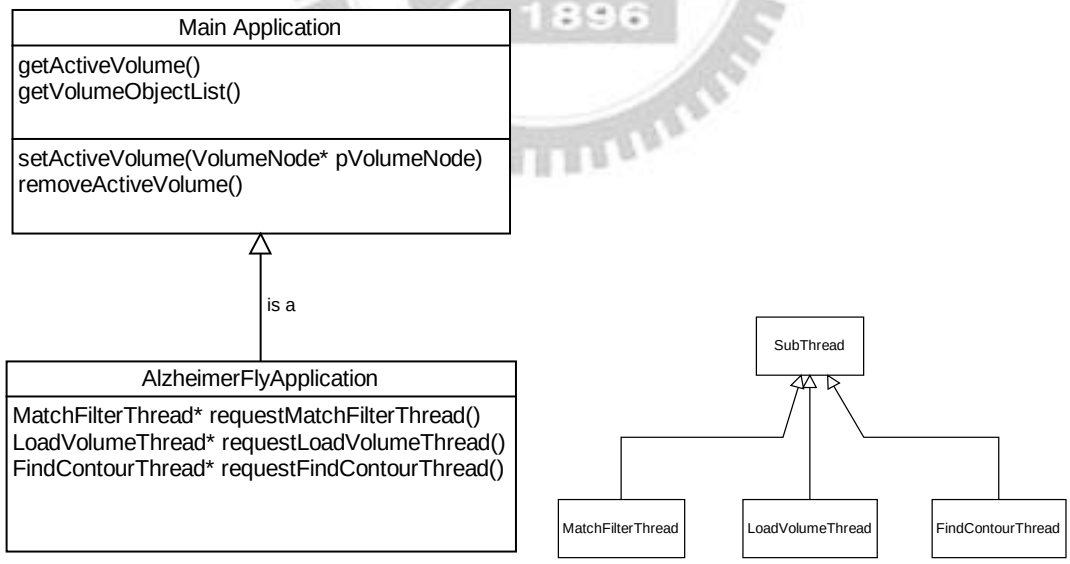


圖 4.9 AlzheimerFlyApplication Class Diagram

AlzheimerFlyApplication 繼承了 Main Application，並且多了三個 methods，這三個看來相當相似，均是提出 Volume Data 的處理程序的需求，

分別為 Matched Filter，Volume Data 之讀取，與作用中 Volume Data 的 Vacuoles 的尋找。一個 vacuole 如[1]，是一個類似球形的空洞結構，因此可能存在數張 slice 當中。而一張 slice 可能存在某一個 vacuole 的切面，稱為 contour 構成，在 FindContour 程序當中即可完成各個 contour 之連結以建立各個 vacuole。

程序執行的機制其實都是相同的，差別在於傳入的參數依據功能有所不同，程序完成後的動作也會有所不同。例如 LoadVolume 會在 MainApplication 中加入一組 VolumeNode，而 MatchedFilter 則會在作用中的 VolumeNode 的子節點加入完成後的 Volume Data。下圖為三個 SubThread 所要傳入的參數

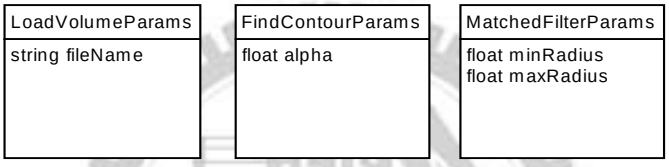


圖 4.10 AlzheimerFlyApplication 各功能的輸入參數

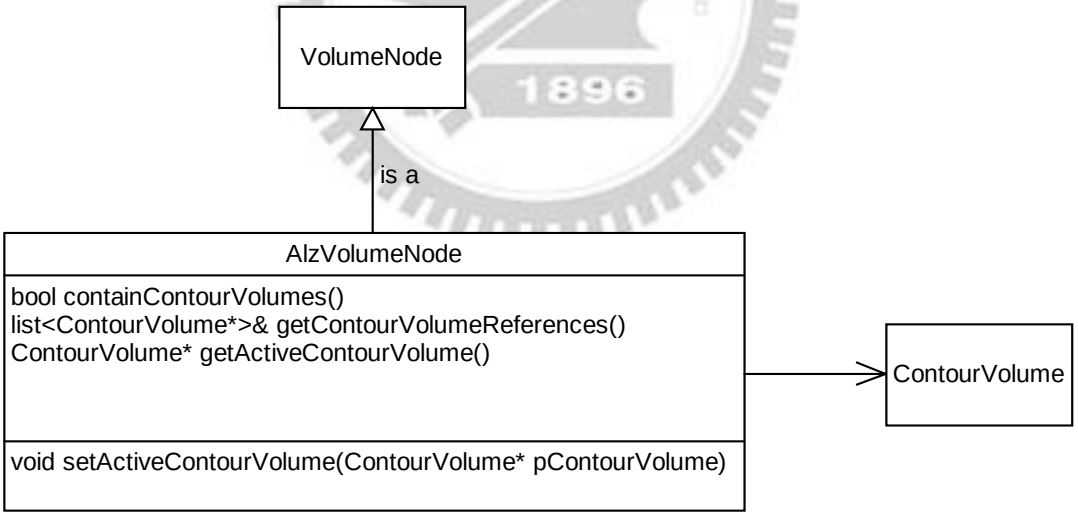


圖 4.11 VolumeNode Specialized for AlzheimerFlyApplication

在 AlzheimerFlyApplication 當中，使用的 VolumeNode 除了具備 MainApplication 當中所述的性質以外，由於 VolumeData 在此應用需求當中，除了影像資料本身以外，仍需要存放 Vacuoles 的資訊。因此我們除了特別設計了 AlzVolumeNode 並將其繼承於 VolumeNode 以外，也在每一個

AlzVolumeNode 存放一組 Vacuoles 的資訊。一組 VolumeData 可能擁有許多張切面(slices)，每張切面如[1]所述，均可能存有數個空洞的位置，而一張 slice 當中，一個空洞以一個封閉多邊形(closed polygon)描述之，而一個封閉的多邊形，又是由數個頂點所構成。這些頂點擁有 x 與 y 的座標，因此從頂點開始，多個頂點(contour vertex)構成一個 vacuole 切面輪廓區域(即 contour)，多個 vacuole(同層切面)構成一個 slice 上面的一組 vacuole 資訊(存成 contour list)，多張 slice 上面的 contour list 形成一組屬於該 Volume Data 的整組 vacuole 資訊(我們稱之 contour volume)，如下圖所述的合成關係(composition)。

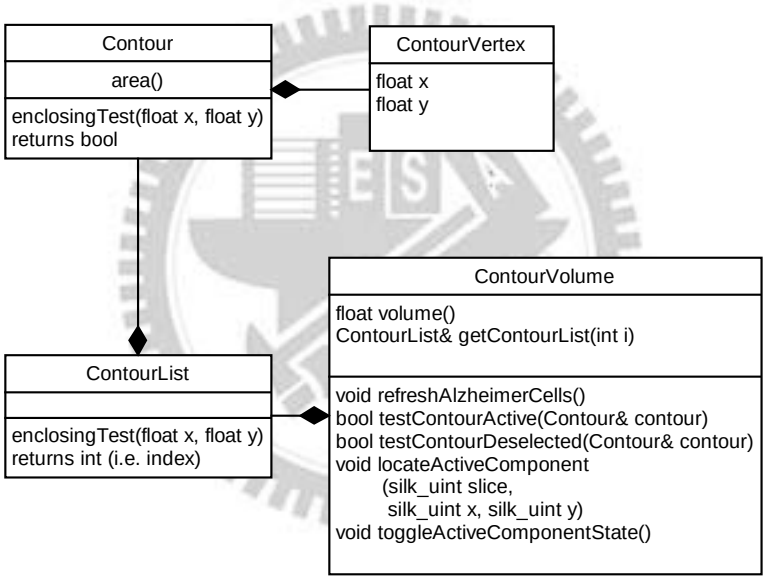


圖 4.12 Composition Relationship of Vacuole Data Structure

以上介紹 vacuole 的資料結構。由於要接收來自使用者的滑鼠點選，以判斷使用者是不是點選於某個 contour 之內，因此對一個 contour(即 vacuole)而言，我們必須提供一個介面，輸入 x 與 y 座標，詢問該 contour 此點是否位在其中。此介面即為 enclosingTest 方法。使用者的滑鼠座標通常會點選於一張切面(slice)上的某個輪廓區域，而一張 slice 則是對應至一組 ContourList，因此一個 ContourList 也提供了 enclosingTest，也是傳入點選的

座標，並搜尋所有的 contours，若有在其中一個輪廓區域內則傳回該 contour 的索引值(index)，否則傳回-1，在 contour volume 當中則將該 contour 設定為”已標示”的狀態，以供 UI 在繪製這些 contour 的時候，能夠將已標示的 contour 以別的顏色畫出。

另外 ContourVolume 提供 volume 屬性，係計算該 volume 當中所有 vacuoles 的體積總和。計算的方法是將每一個 vacuole 的每一個 contour 的面積(area()方法)相加（須扣掉被使用者排除的部分）。使用者可在點選其中一個 vacuole(某個 contour)以後，決定是否排除該 vacuole 體積的計算。排除的動作經由 UI 程式操作 ContourVolume 當中的 toogleActiveComponentState 方法即可切換目前是否列入排除的 vacuole。

以上是 AlzheimerFlyApplication 的設計部份，下一小節我們介紹**半自動果蠅腦神經追蹤系統**在 MainApplication 當中的設計。

4.2.3 應用需求：半自動果蠅腦神經追蹤系統－NeuTrApplication

對此應用需求而設計的 class 命名為 NeuTrApplication，並繼承於 MainApplication。下圖為 NeuTrApplication 的 class diagram:

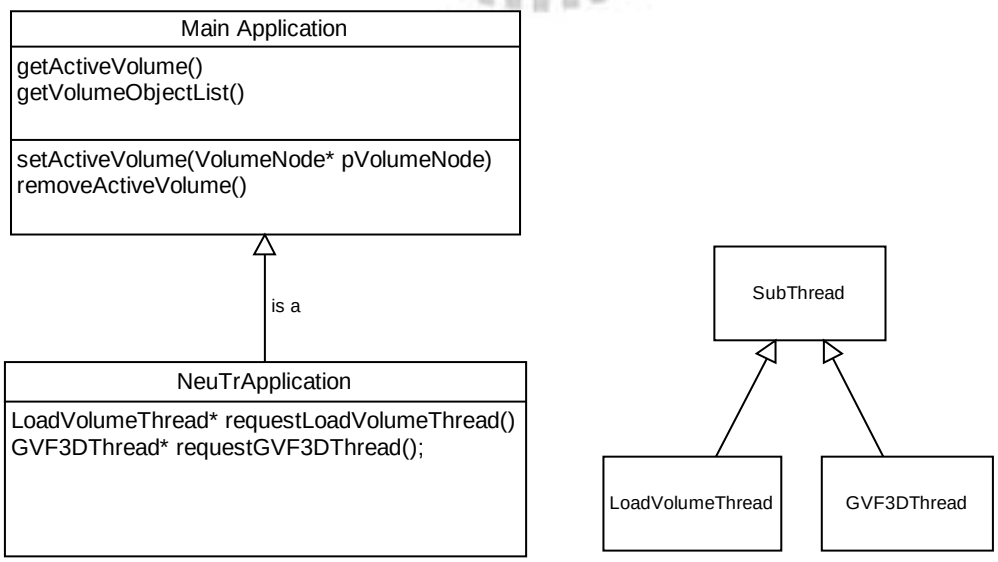


圖 4.13 NeuTrApplication Class Diagram

NeuTrApplication 繼承了 Main Application，並且多了二個 methods，除了 LoadVolumeThread 與 4.2.2 所述相同以外，GVF3DThread 是在本應用需求當中最為主要的處理程序，這是計算一組 VolumeData 的 GVF 所要用的。

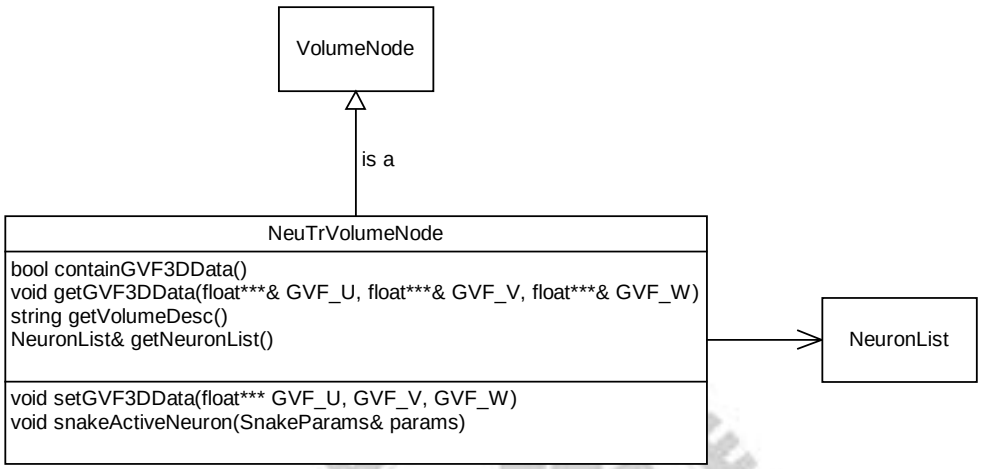


圖 4.14 VolumeNode Specialized for NeuTrApplication

在 AlzheimerFlyApplication 當中，使用的 VolumeNode 除了繼承於 VolumeNode 以外，尚需要存放 GVF 的計算結果，與使用者所編輯的 Neuron 資訊。如上圖，一組 VolumeData 將會含有一個 NeuronList 以存放一組 Neuron。一組 Neuron 由多個(0..*)Neuron 所構成，在本系統當中經由使用者編輯而成。一個 Neuron 以多個點(3D)形成的 polyline 表示，每個點擁有 x、y 與 z 座標，分別表示該組 VolumeData 的 slice z, row y, 的像素 x。

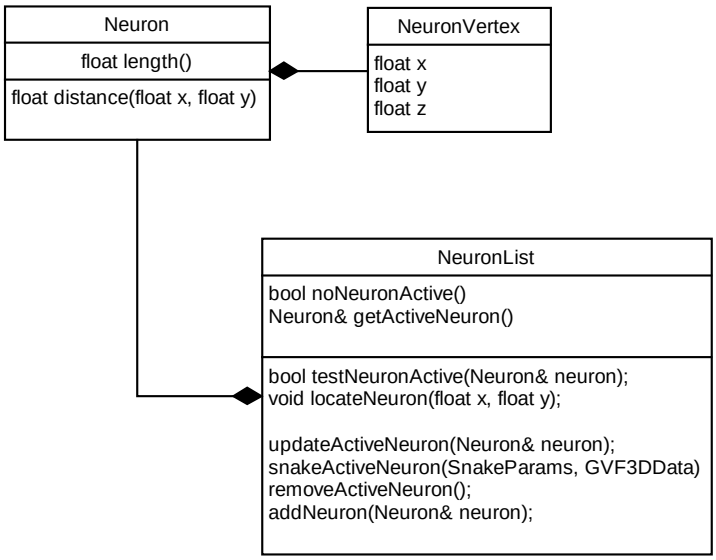


圖 4.15 Neuron Data Structure

以上介紹 Neuron 的資料結構。由於要接收來自使用者的滑鼠點選，以判斷使用者點選的位置是否靠近該 Neuron(約 5-7 pixel 以內)，因此對一個 neuron 而言，提供 distance 方法(method)，輸入 x 與 y 座標，詢問該 neuron 此點是否位在其中。由於 neuron 各點都是三維的，但以正交投影(Orthogonal Projection)至螢幕上以後，是平面的，因此只取 x 與 y 座標即可。另外在 NeuronList 當中，我們亦提供 locateNeuron 的方法，同樣傳入滑鼠點選的 x 與 y 座標，此時 NeuronList 則從 neuron 當中找出與點選的座標最靠近（且足夠靠近約 5-7 pixel 以內）的 neuron 以提供 User Interface 判斷是否為作用中的 neuron，並以不同顏色標示。對於作用中的 Neuron，亦可使用 NeuronList 提供的 snakeActiveNeuron，以 GVF 資訊輔助，對其以 snake 演算法變形，使得能夠更貼近原 Volume Data 之特徵之位置。

另外透過 NeuronList，我們亦可從 getActiveNeuron 取得目前的 Neuron 在 UI 當中編輯（例如在 Neuron 的末端加入一些節點），並以 updateActiveNeuron 更新作用中的 Neuron 資訊，或是以 removeActiveNeuron 移除作用中的 Neuron。至於 addNeuron 則是傳入一個 Neuron 以在 NeuronList

加入一個 Neuron。其中，對於 Neuron 的加入與修改等動作，均由使用者以滑鼠點選編輯而成，因此我們為 Neuron 以 Builder Pattern[9][10]設計了一種建構程序：

NeuronBuilder – 一種 Builder Pattern 的實作

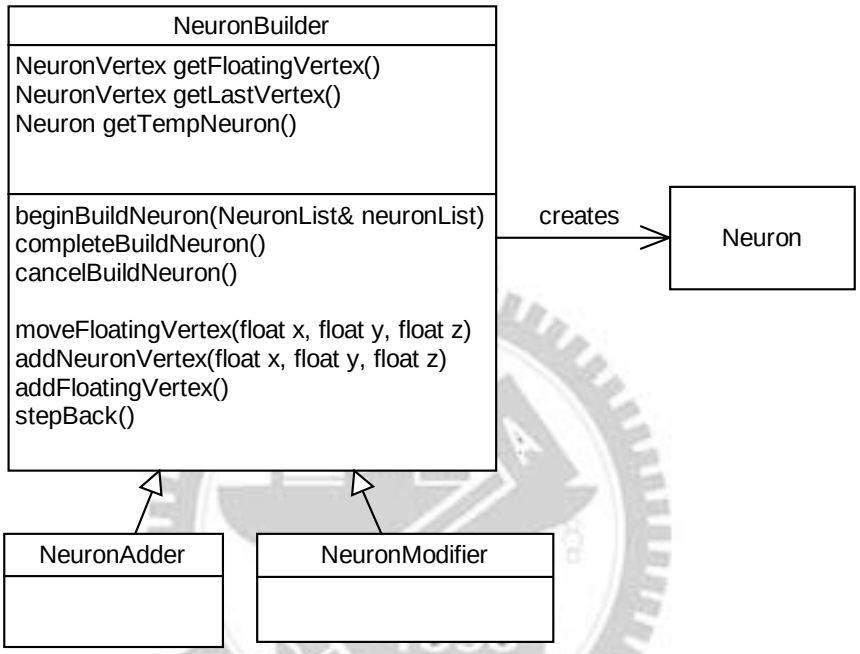


圖 4.16 NeuronBuilder Class Diagram

NeuronBuilder 主要提供 Neuron 新增與修改的功能，又分為用來新增一個 Neuron 的 NeuronAdder 與修改作用中的(即被標示出的)Neuron 的 NeuronModifier。NeuronBuilder 的執行狀態表示如下：

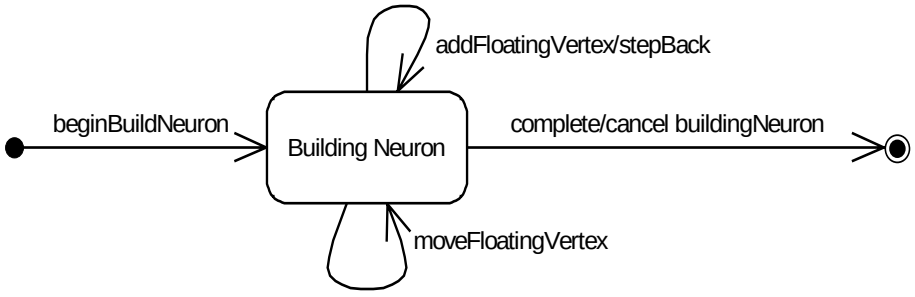


圖 4.17 Building a Neuron

NeuronBuilder 均按照此方式實現 Neuron 的建立與修改。執行

beginBuildNeuron 方法並傳入目前作用中 Volume Node 的 NeuronList 以開始建立或修改 neuron，這時候若使用 NeuronAdder，此 builder 會存放一個不包含任何頂點的 Neuron，稱為 tempNeuron；若為 NeuronModifier，其 tempNeuron 則為作用中的 NeuronList 的目前所標示的那個 Neuron 的 reference。

除了 tempNeuron，也存在一個用來存放即將加入的 floatingVertex。在 UI 中，若使用者將滑鼠移動到影像中的某個座標，且這時候正在編輯 Neuron，則會將 floatingVertex 的 xy 與 z 分別設定為使用者的滑鼠座標(x 與 y)以及目前瀏覽的切面索引(z)。若移動到某個頂點接之後使用者確認將此點加入，則將 floatingVertex 加入 tempNeuron 當中。使用者反覆的重複此動作，直至完成編輯之後 NeuronBuilder 將 tempNeuron 加入或更新於 beginBuildNeuron 時所傳入的 neuronList。

以上介紹 MainApplication 的設計，根據兩種應用需求分為 AlzheimerFlyApplication 與 NeuTrApplication。其中 AlzheimerFlyApplication 提供 MatchedFilter 的計算、Vacuoles 的搜尋，並存放找出的 Vacuoles 以供使用者點選是否列入計算，與計算該 VolumeData 當中的 Vacuoles 總體積。而 NeuTrApplication 則提供了便利 UI 程式將使用者編輯的 Neuron 加入／更新至其中的 NeuronBuilder，與 GVF 的計算程序以輔助 Neuron 的 Snake 變形演算法。

在下一章中，我們將介紹使用者介面的系統設計，並提供一些使用範例的展示。

第五章 使用者介面系統設計與系統使用實例展示

在 3.2 的 Main Package Diagram 當中， User Interface 元件位於本系統中的最上層，亦即最為接近使用者。在本章中，我們將介紹本系統當中使用者介面的設計部份。除此之外，我們也針對這兩個應用需求，提供一些使用實例作為展示。5.1 的部分介紹使用者介面系統的兩個主要設計模式，包含 Model/View/Controller 與 Qt 的 Library 所提供的 signal/slot 設計模式。5.2 則是針對兩個不同的應用需求，提供使用實例展示。

5.1 Model/View/Controller 與 Signal/Slot 設計模式

在使用者介面的設計當中，最重要的問題在於使用者、各個視窗元件與系統本身三者關係如何互動。其中，Model/View/Controller[9]將系統分解成存放、呈現與操作三個部分，是一種已被廣泛用於視窗式介面程式的設計模式，我們在 5.1.1 節介紹之；另外一種稱作 Signal/Slot，是 Trolltech 公司在產品之一－視窗設計函式庫 Qt 當中所提供的一種設計模式，比較適合用於視窗中物件與物件之間的事件傳遞。此方法是在某個物件定義發出的事件(signals)，發生以後由另外一個或數個物件所定義的接收端(slot)接收並處理相關的命令，我們將在 5.1.2 節介紹之。

5.1.1 使用者介面與 Model View Controller 設計模式

在 3.2 提到的 User Interface 元件，最主要的工作是建立一個視窗式介面系統，供使用者經由鍵盤與滑鼠，以圖形化的方式操作本系統，並能瀏覽所處理之影像資料與各系統功能之執行結果。使用範例將於 5.2 中介紹。本系統的 UI 依照 Model/View/Controller 設計模式[9]實作而成，我們在此節將介紹此一設計模式。

下圖介紹本系統中的 User Interface 元件之主要成員：

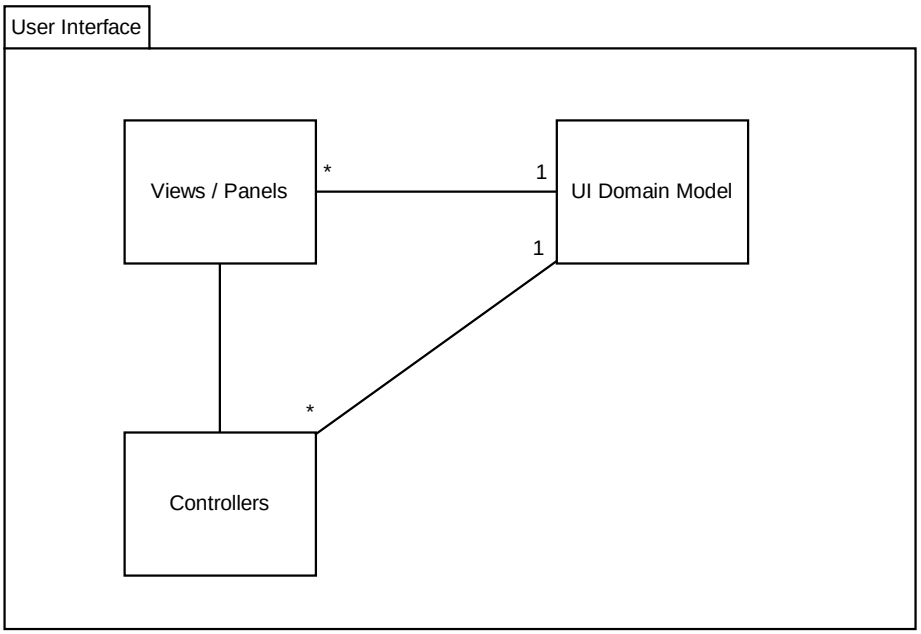


圖 5.1 Entity Diagram of UI Package

User Interface 元件之組成－Model/View/Controller：

本元件－User Interface 由三種不同的物件所組成，分別為視圖（或面板）、控制與模型，也就是典型的 Model-View-Controller (MVC)設計模式[9]。本設計模式通常用於 GUI 程式設計。相對於傳統的電腦程式之輸入－運算－輸出之執行模式，在 GUI 的程式當中，輸入、運算、輸出分別由 Controller、Model、View 來完成：使用者經由圖形式介面下達指令，控制器(Controllers)將這些指令直譯(interpret)成某種系統提供的功能，並對於模型(Model)執行其命令。完成命令後，若有執行結果，或是模型當中的資料有所改變，則會發出事件(Event)，通知視圖(View)更新顯示資訊。視圖可經由存取模型的資料，轉換成易於理解的圖像或是文字顯示於螢幕上以供使用者瀏覽。

Model/View/Controller 之本體(Entity)關係：

在本系統中，Model/View/Controller 的本體(Entity)關係如圖 5.1 所述，View 與 Model 為多對一，也就是多個視圖共同接觸同一 Model；View 與 Controller 為一對一，意思是說，一個 View 經由所屬之 Controller 傳遞事件，因此也能得知 Controller 與 Model 亦為多對一的關係。

View(視圖)與 Panels（控制面板）：

View 主要的功能，是將存放於 Model 的應用程式資料(Application-specific Data)以易於理解的方式呈現予使用者，並接收來自使用者的指令（滑鼠或鍵盤等），例如圖 5.6 中數字 3 所指示的視窗。然而，有些視窗並不需要將任何資料呈現，卻提供了一些按鈕或是供使用者操作的圖形物件（如右圖）。因此在本系統中，我們將這類視窗物件與 View 分別，命名為 Panels－「控制面板」。

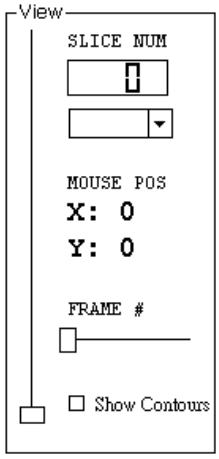


圖 5.2 View Panel

Panels 屬於使用者介面的一部分，提供使用者一系列的控制介面，以使用系統的所有功能。舉例來說，一組 Volume Data 可能有許多張影像疊合而成，因此需要讓使用者控制目前顯示的影像（例如第幾層也就是 slice）。右圖也就是圖 5.6 中數字 4 所指示的視窗，當使用者移動了任何一個控制棒(Slider)，則顯示的影像將會切換到相對應的層面(slice)。像這樣的元件在此稱作 Panel(面板)。

使用者介面的 UI Domain Model：

本系統中，除了 View 與 Controller 以外，Model 的部分，命名為 UI Domain Model。在傳統的 Model/View/Controller 設計模式中，Model 主要扮演存放系統的資料與狀態，並提供一組介面供存取與功能執行或是資料處理之用。但在本系統的 User Interface 當中，Model 的部分除了儲存系統的所有資料，例如影像資料（藉由第二層的 Main Application 存取）以外，另外亦存放與 UI 相關的資訊如系統功能進行的狀態，並協助 Main Application 在系統資料有所變更以後發出相對應的事件以通知各個視窗元件更新顯示。因此我們將 Model 的部分以 UI Domain Model 命名之。

User Interface Package 各物件之存取關係(Association)：

下圖表示在 User Interface Package 當中的各個物件之存取關係圖：

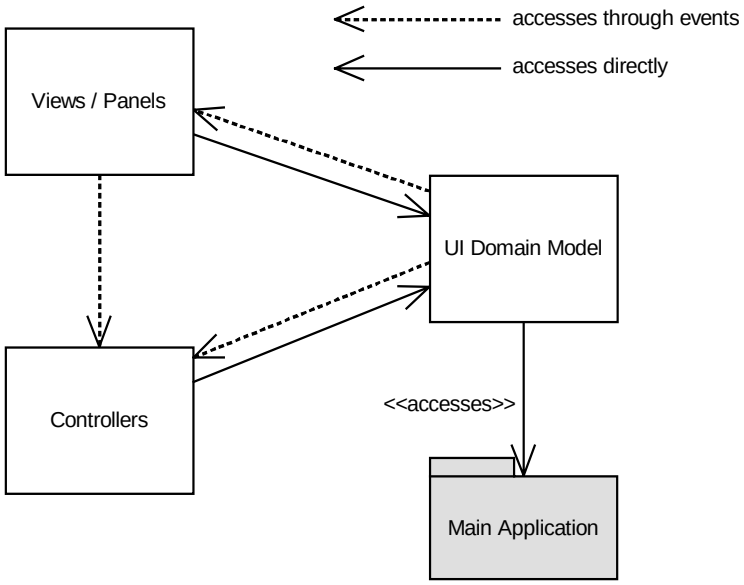


圖 5.3 UI 之各物件存取關係圖

從圖中可以看出 Views / Panels、UI Domain Model、Controllers 等物件之存取關係。實線所表示的部分為直接存取，也就是該物件擁有存取對象的 reference(參考)，可直接操作該物件的所有屬性(attributes)與方法(methods)；虛線則表示該物件並不擁有存取對象的 reference，但是可以透過事件傳遞讓其接收，並執行適當的操作。

Model 如何與 View/Controller 溝通

在 Model/View/Controller 設計模式當中，Model 主要的工作為存放與處理系統資料，並提供 View 物件查閱之、與 Controller 物件經由 Model 提供的 Methods 操作之。因此 Model 本身並不需要了解 View 與 Controller 之實作，也不需要對此兩物件直接操作。然而，遵循此設計模式，當 Controller 對於 Model 的資料有所操作之後，應當通知與該 Model 的所有相關的 View 物件更新其顯示資訊。因此，若要讓 Model 能夠主動的與 Views 或 Panels 溝通，我們將在 Model 上定義所發出的事件，以通知 View 或 Panel 該 Model 資料已經被更改的訊息。下圖表示一個更新應用程式資料的動作，Model/View/Controller 三者之互動情形：

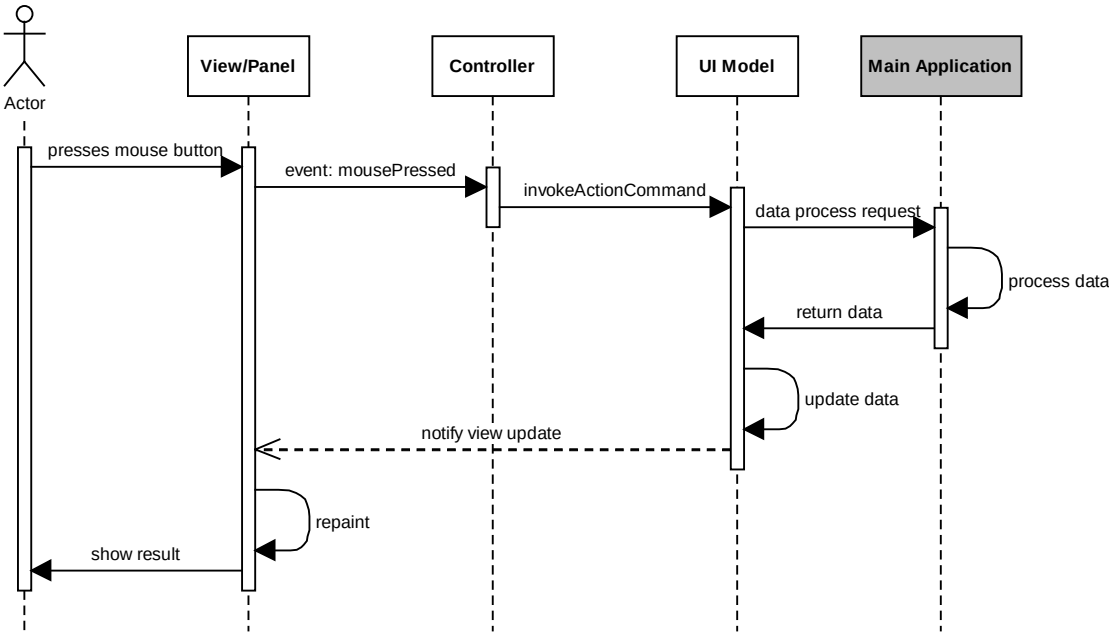


圖 5.4 Model/View/Controller Sequence Diagram

以本系統為例，1.當使用者從圖 5.2 的視窗（此定義為 View 物件之一）當中，移動一下調整桿(slider)，2.其所屬的 Controller 將這樣的事件解讀(interpret)成切換至某一層(slice)的影像，3.並命令 Model 更新目前使用者瀏覽的影像之索引值(index)。4.Model 更改了索引值，並做了一些相關處理（例如更新暫存區的影像資料）之後，引發出「顯示資料已經被變更」事件。5.該事件傳至主程式之後，由顯示影像的視窗(View)接收之後，更新顯示影像的畫面。

5.1.2 Signal/Slot 設計模式

在本系統的Model/View/Controller當中，UI Model扮演資料存放與操作的角色，供View的讀取和Controller的控制。然而，UI Model實際上並不知道View與Controller的存在，因此需要藉由事件傳遞至相關的視窗以通知資料的更新。很幸運地，Trolltech®的Qt Library已為我們提供這樣的設計模式，我們可在UI的視窗中，定義事件與接收端(signal/slot)表示這個視窗將會發生哪些事件與接收哪些事件。圖示如下：

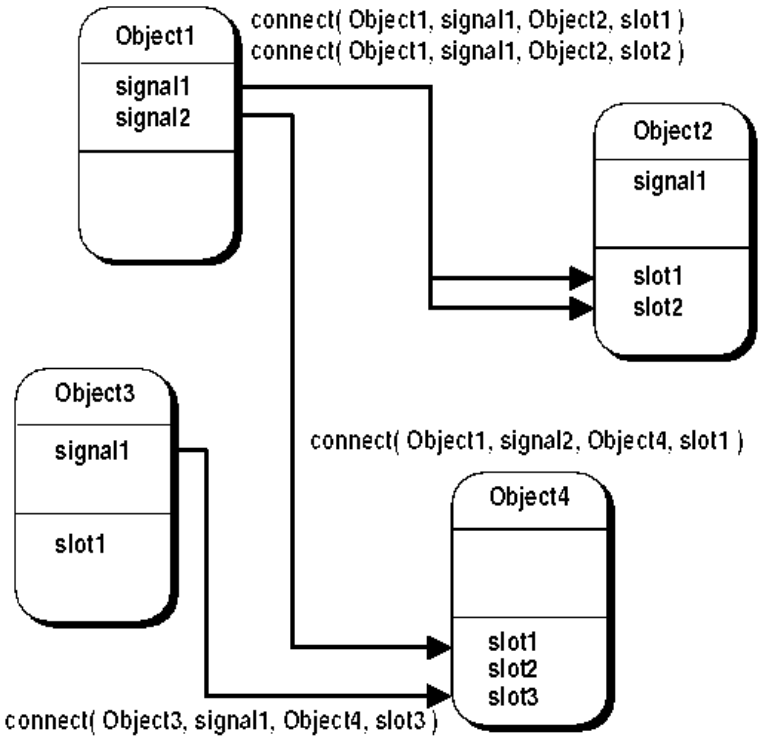


圖 5.5 Illustration of Qt Signal/Slot Mechanism

一個 Object(可能為視窗或是一般物件)當中，包含 signal 與 slot 兩部分，表示這個物件可發出或是接收哪些訊息。這些 signal 與 slot 在 Qt 當中可寫成一般的函數，而發送與接收端的 signal/slot 函數傳入的參數相同即可連結。在本系統中除了用來將 Model 的訊息傳至 View/Panel 以外，對於不需要操作至 Main Application 層級的一些簡單的指令(見 3.2)，也可用 Signal/Slot 機制直接在視窗與視窗之間傳遞。例如使用者在 Image View(顯示影像的視窗)當中移動滑鼠游標時，Image View 會發出訊息並附上新的滑鼠游標，傳至 View Panel 的 slot 以更動目前顯示的滑鼠游標之座標。

5.2 使用者介面系統展示

在上一節裡，已提到 Model/View/Controller 設計模式。在 5.2.1 節當中，我們將介紹 UI 程式的主畫面與其當中各個視窗元件。在 5.2.2 節，我們將從設計的角度，介紹系統當中各視窗與系統中各項功能的互動機制。

5.2.1 使用者介面之主畫面

在展示本系統的過程中，我們以Qt的Library實作使用者介面系統，並使用MinGW作為C++程式碼的編譯器。我們使用Microsoft® Windows XP作為我們的執行平台，由於本系統常常需要讀取Volume Data這類的三維影像，記憶體的需求量通常較大。對一組 64 張slices的 512 x 512 的果蠅腦神經影像之GVF資料而言，對每一個voxel儲存一個三維的向量，而每一分量的資訊需要以 4 個位元組的floating point變數(C++ data type based on 32-bit computers)儲存，因此需要 64 x 512 x 512 x 12 bytes = 192MB的記憶體儲存。

我們在展示本系統所使用的硬體環境如下：

CPU: Intel Pentium IV 2.4GHz Processor

Memory: Transcend 512MB DDR400 (2pcs, totally 1GB)

Hard Disk: 160GB Western Digits

下圖為本系統的主畫面範例：

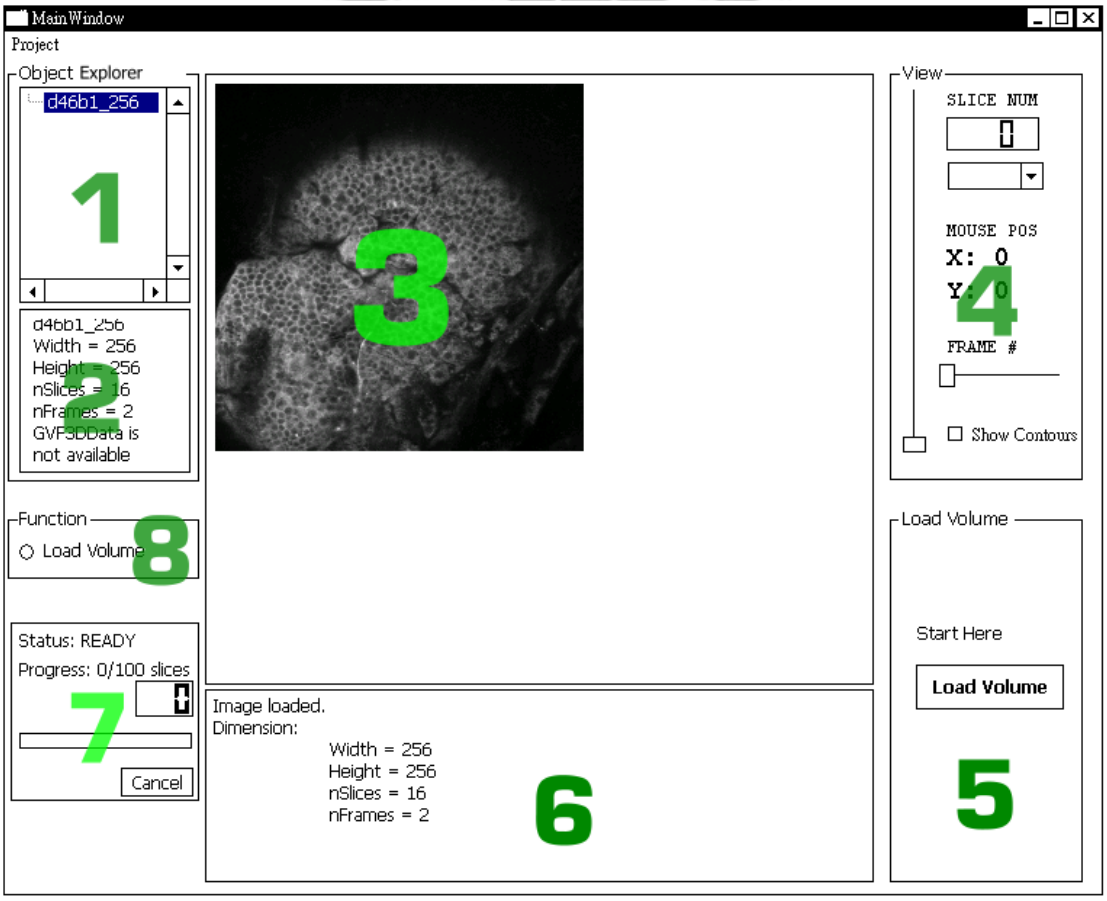


圖 5.6 使用者介面範例

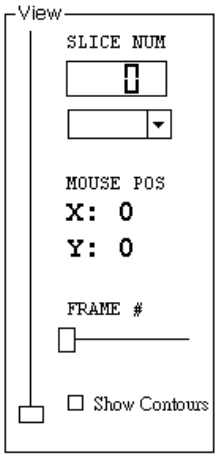
本系統的使用者介面之各個視窗在 UI 的 MVC 當中均屬於 View/Panel 的部分。接下來我們將逐一介紹主要的視窗元件：

1. 影像物件瀏覽視窗(Object Explorer)：使用者在此視窗中，可以樹狀結構的方式，觀看到目前本系統中的所有影像物件(Volume Data)。使用者可從這些物件當中點選有興趣的物件以觀看該影像資料。當使用者點選以後，該物件將會成為本系統目前作用中的 Volume Data，執行 MainApplication 當中的所有功能時，均會對其發生作用。

2. 影像物件資訊視窗(Object Explorer)：當使用者從(1)點選一組影像物件以後，此視窗會顯示該影像物件的相關資訊，例如該 VolumeData 的 Dimension 等。

3. 影像瀏覽視窗(ImageView)：該視窗除了顯示使用者所點選的影像物件資料以外，也能夠接收使用者對於該影像的編輯命令。例如在 NeuronTracing 當中，使用者可從影像當中直接在某個 Neuron 附近點選，以切換目前作用中的 Neuron。

4. 影像瀏覽控制器(ViewPanel)：右圖中為該視窗之外觀，主要提供使用者控制目前瀏覽的影像位置。由於影像資料均以 Volume Data 為主，因此觀看的時候需要指定目前觀看的切面(slice/frame)位置。而 SLICE NUM 則顯示目前觀看的 slice 之 index。數字下方的一個類似選單的介面則是提供使用者觀看影像時調整影像的比例，可放大或縮小影像。另外 MOUSE POS 則是當滑鼠移動至影像(3)當中，顯示滑鼠位於影像中的像素實際(pixel)位置。這樣的數值不會受到放大／縮小瀏覽影響。最後 Show Contours 的切換器，則是供使用者決定是否觀看影像以外的資訊，例如使用者編輯中的 Neuron 或是系統找出的 Vacuole 的 contour 資訊。



5. 功能使用介面(FunctionPanel)：使用者從此處執行系統提供的各項功能(即 MainApplication 當中所提供的例如 Matched Filter、Vacuole Search 和 GVF 的計算，或是將 Neuron 以 Snake 變形)，圖中的範例的 Load Volume 是從檔案中讀取一組 Volume Data 成為影像物件並加入本系統。使用者能夠以視窗(8)的功能列表切換至想要執行的功能。

6. 訊息視窗(MessageOutput)：系統使用中若有任何訊息均顯示於此，通常用以通知使用者某個處理程序的執行時所發生的事件，例如開始執行、被使用者終止、執行過程中發生任何問題或是執行完成等，均會提供文字訊息顯示於此。

7. 處理程序執行狀態(ThreadStatusPanel)：當使用者對於系統發出處理程序（例如 Matched Filter 演算法或是計算 GVF）的要求時，系統將會開始執行處理程序。當執行時間較長而需要等待時，處理程序的進度說明與完成的百分比均在此顯示；文字的部分則是目前處理程序的狀態（見 4.2.1 的 SubThread）另外顯示 cancel 的按鈕則是供使用者終止目前正在執行的處理程序。

8. 功能切換器(FunctionListBox)：當使用者點選某個影像物件時，可用的功能會隨著作用中影像物件的屬性而有所改變。例如，使用者在應用需求 1（找尋 vacuole）當中，若點選未經處理過（即是剛從檔案輸入）的 VolumeData，可對其計算出 Matched Filer 之後的 Volume Data；對於 Matched Filter 計算的結果影像物件，則提供尋找 Vacuoles 的功能。當使用者從切換器當中選擇了一項功能以後，視窗(5)將會切換成對應的功能表以供使用者設定適當的參數並執行功能。

9. 主視窗(MainWindow)：包含所有的視窗元件，並控制各個視窗元件的訊息連結(signal/slot)。主視窗也含有一個稱為 UI Domain Model 的物件與數個 Controller(5.1)，完成各視窗元件與系統之間的連結。

10-14 均屬於 FunctionPanel，顯示於視窗(5)的位置：

10. 影像讀取功能表：即畫面中的(5)，僅有一個按鈕。當被按下之後，會要求使用者指定檔案名稱供系統讀取該檔案資料以建立影像。

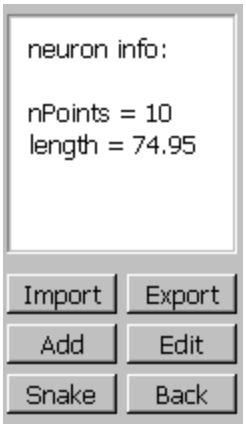
11. Matched Filter 功能表：屬於應用需求 I 的設計。提供一個數字欄位，供使用者設定執行 Matched Filter 時所要用的 Mask Size，以及啟動程序的按鈕。

12. Find Contour 功能表：屬於應用需求 I 的設計。提供二個數字欄位，供使用者設計執行尋找 Vacuole 演算法所要用的參數，分別為 Alpha Shape 的 Radius 與構成一個 Contour 所需要的最少的 Vertice 數。

13. GVF 功能表：屬於應用需求 II 的設計。提供各項 GVF 參數輸入的欄位。.

14. NeuronEditor 功能表：屬於應用需求 II 的設計。如圖：

使用者可從 Import/Export 將目前現有的 Neurons 讀取／儲存至檔案中；Add 與 Edit 分別為 Neurons 的新增與編輯的功能；Back 按鈕則是在 Add/Edit 當中作用，是刪除編修中的 Neuron 最末節點。若使用者從影像中選取一個 Neuron 之後，則會顯示與該 Neuron 有關的資訊，例如有多少個點與總長度。



5.2.2 系統使用實例展示

在 5.2.1 當中，我們已介紹了本系統的主畫面。對這兩項應用(找 Vacuole 與 NeuronTracing)而言，外觀上大致來說是相同的。我們在此節中將會個別針對這兩項應用介紹本系統使用上的畫面實例：

A. AlzheimerFlyApplication

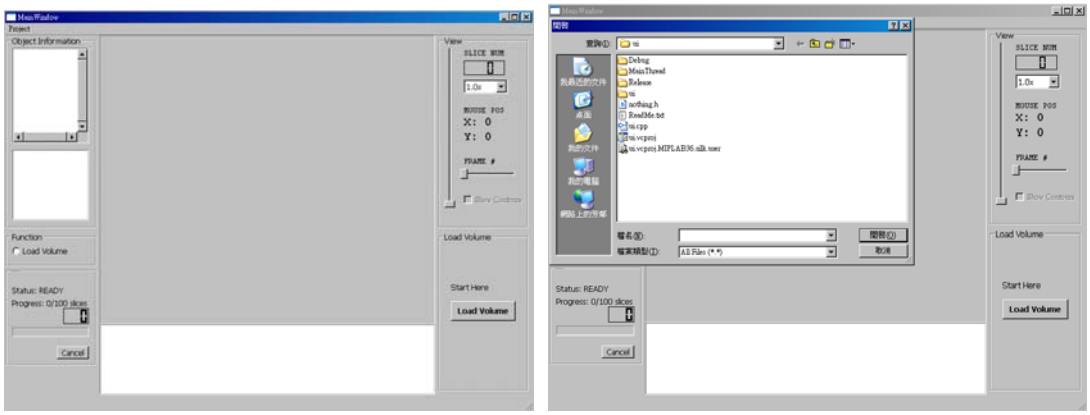
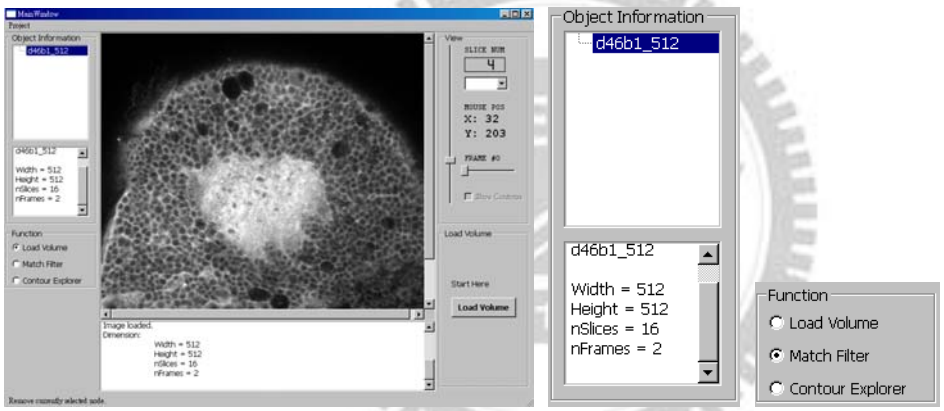


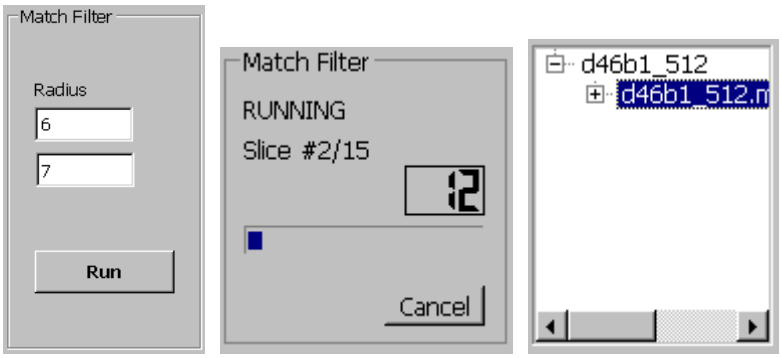
圖 5.7 AlzheimerFlyApplication Demonstration

(a)使用者介面初始畫面　　(b)按下 LoadVolume 以後出現的視窗以供使用者選擇檔案



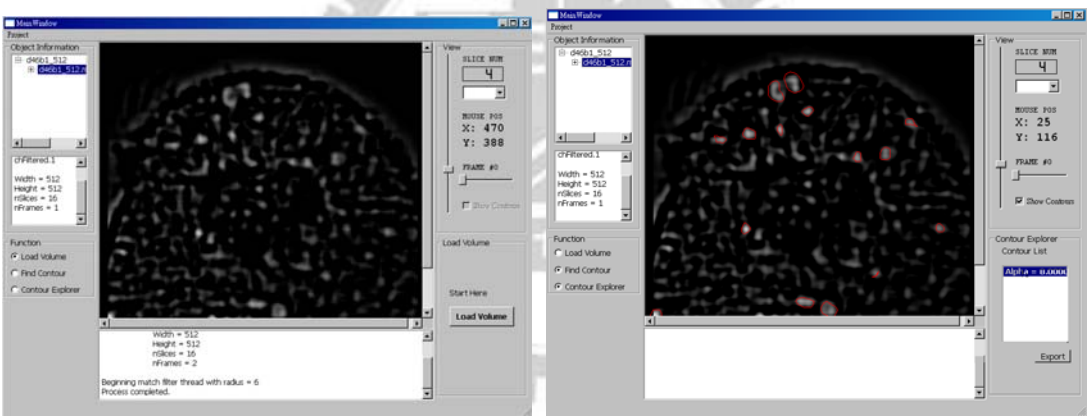
(c) 讀取影像之後的畫面　　(d) 影像物件瀏覽視窗與功能表

完成讀檔的動作之後，在左上角(1)的視窗中，會出現影像的名稱(以檔案名稱命名)，並顯示影像的相關資訊。以此為例，是一組 16 張 512x512 的影像，共 2 個 volume。經由使用者從 ViewPanel 或是使用滑鼠滾輪在影像視窗中切換之後，目前顯示的是 slice 4 的影像。而此時 ViewPanel 當中由於使用者已將滑鼠移到影像視窗中而顯示了目前滑鼠於影像的座標位置。另外，由於影像已被選取，在(d)當中的功能表已加入了該影像的相關功能，因此可執行 Matched Filter 演算法。



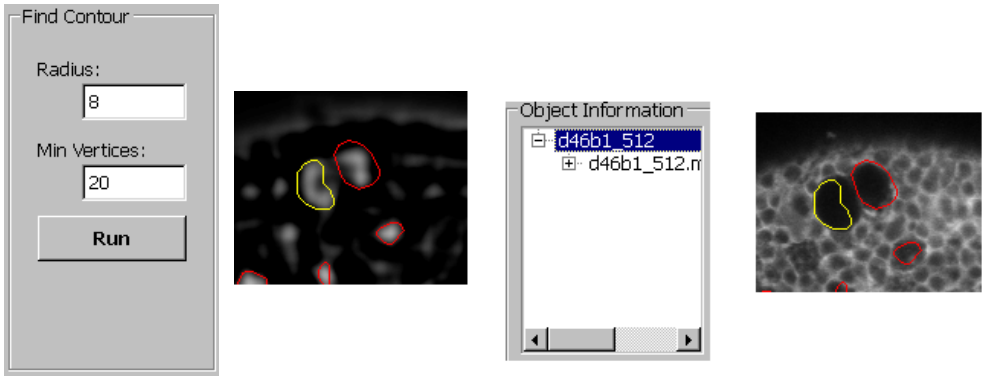
(e) Matched Filter 功能表、處理程序進度顯示、完成後的物件瀏覽視窗

選取了 Matched Filter 以後，右下方出現了 Matched Filter 的功能表，使用者填入了適當的參數之後(尋找半徑為 6-7 像素的圓)，按下了 Run，畫面的左下方則出現了目前處理程序的進度，以上圖為例，目前正在處理 slice 2，完成度為 12%。完成之後，影像物件瀏覽視窗在原先的 Volume Data 下方出現了一組影像，是完成 matched filter 之後的結果。



(f) 完成 Matched Filter 之後顯示的影像、找出的 Vacuole 於 slice 4 之切面

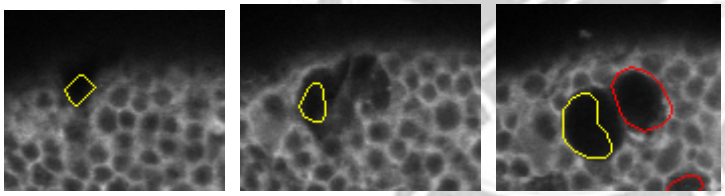
完成 Matched Filter 之後，圖(f)左方顯示的是 Matched Filter 之後的 slice4 之影像，選擇完成 Matched Filter 影像始能進行 Vacuoles 的尋找，即可使用 Find Contour 功能表。如圖(g)，使用者填入參數之後按下執行，待進度完成後即出現上圖右方的畫面，紅色的圈圈代表所找到的 Vacuoles 在 Slice 4 當中的切面。



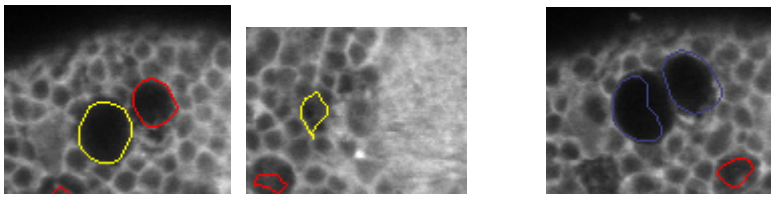
(g) 切換至原圖並顯示找到的 Vacuole 之切面

使用者若在某個紅色圈選的區域內點選，該邊界會變為黃色，也就是成為作用中的 Vacuole。另外各層 Slice 之間與該 Contour 連結的(屬於該 Vacuole 的)Contour 均會變為黃色。如圖(h)顯示 slice2 至 slice6 均被選取。

使用者可從本系統介面之左上方的影像瀏覽視窗中選擇至原圖，此時因為已經完成了 Vacuoles 的搜尋，而能在原圖當中看到 Vacuole 的切面以供使用者判斷是否真的為 Vacuole。若不是正確的 Vacuole 則能夠選擇將其排除以不列入 Vacuoles 體積的計算。



(h) slice 2　　slice 3　　slice 4



slice 5　　slice 6　　slice 4-deselected

Total vacuole volume: 24246
Total vacuole volume: 24144

(h) 選取中的 vacuole 在 slice 2 至 slice 6 的位置與形狀的變化情形

使用者點選了某張 slice 上面的一個紅色邊框的區域之後，即選取了屬於該 contour 的 vacuole，因此該層 slice 附近屬於該 vacuole 的 contour 均會

被選取。於是 slice 2 至 slice 6 該 vacuole 的所有切面均成為黃色。另外，若使用者在選取中的區域內連續點選兩次，則能夠切換該 vacuole 是否列入所有 vacuole 總體積之計算。若有個 vacuole 被排除計算之外，則該 vacuole 的所有 contour 會呈現紫色。每次切換後，總體積會在訊息窗當中顯示出。

B. NeuronTracing

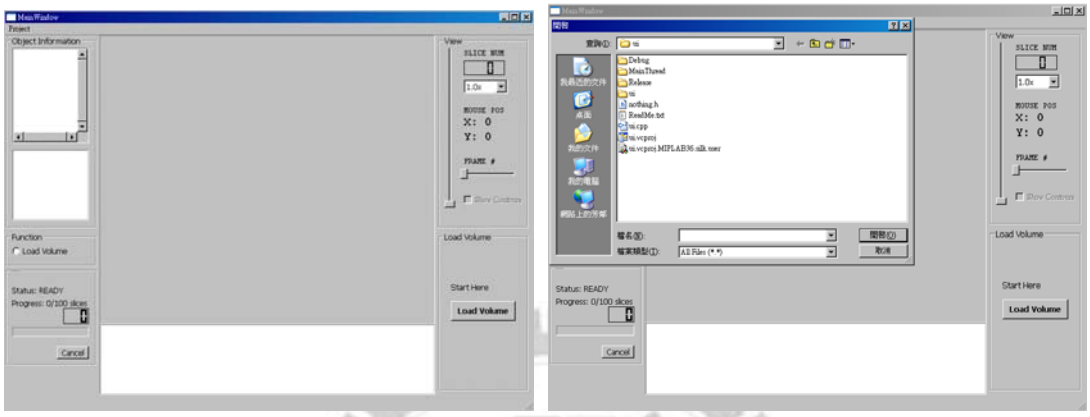
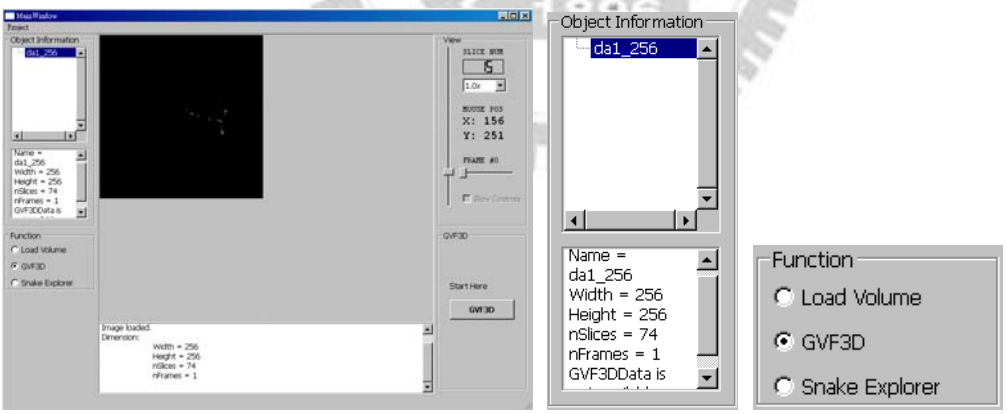
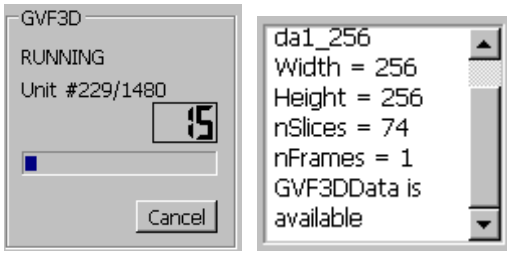


圖 5.7 NeuTrApplication Demonstration

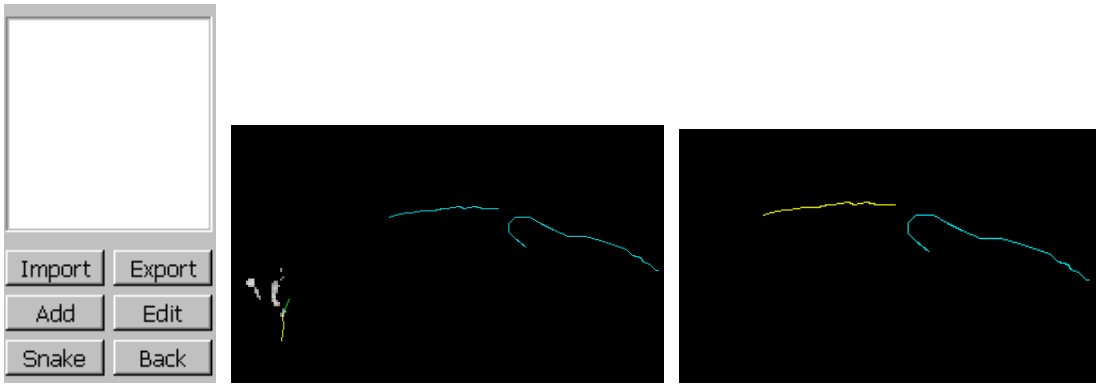
(a)使用者介面初始畫面　　(b)按下 LoadVolume 以後出現的視窗以供使用者選擇檔案



(c) 讀取影像之後的畫面　　(d) 影像物件瀏覽視窗與功能表



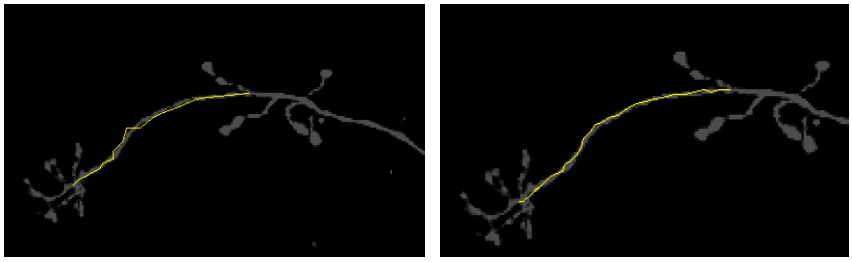
(e) 執行 GVF 的計算，完成之後畫面顯示 GVF3DData is available 表示計算完成，該 Volume Data 擁有 GVF 的資料，可開始編輯 Neuron。



(f) 正在編輯 Neuron，右圖為完成編輯之後，選取其中一個 Neuron 而變成黃色。

使用者在完成 GVF 的計算之後，可經由按下 NeuronEditor 功能表中的 Add 新增一個 Neuron，即可以滑鼠點選的方式並配合滾輪切換各層的 slice 完成編輯。以上圖為例，使用者已完成編輯兩個 Neuron，並在左邊的 neuron 附近點選之後，成為被選取的 neuron 而顯示為黃色。選取以後按下 Edit 之後即可編輯，也就是在 neuron 之後延伸別的節點，或是在編輯模式當中按下 Back 以刪除最末端的節點。若在編輯中放棄更改，則按下滑鼠右鍵即可。

由於這些 Neuron 均為手動編輯而成的，難免對於真實的位置有些誤差，因此對於完成編輯的 Neuron，能夠以 snake 演算法進行變形的動作。使用者選取想要調整的 Neuron 之後按下 snake 將會出現一個對話方塊提供參數的設定，確定之後即可完成調整的動作。下圖中分別為調整前後的 Neuron，很明顯的可以看見調整過後的 Neuron 更為接近 Neuron 的邊線。



(g) Snake 調整完成編輯的 Neuron (before and after)

第六章 結論與未來展望

目前我們已對於共軛焦顯微鏡的影像分析工作，提供了一種使用者介面系統的設計，並針對兩種不同的應用，實作出兩組使用者介面軟體。此兩應用分別為**果蠅腦之老年痴呆情形分析**與**半自動果蠅腦神經追蹤**，並在第五章介紹了使用上的實例展示。

在進行我們的實作階段中，其中比較需要關切的是記憶體管理的問題。目前若使用我們的系統，配合 1GB 的主記憶體，計算資料量較大的 Volume Data 之 GVF，例如 64 張 1024 x 1024 的影像，則會因為記憶體不足而無法執行程式。關於這點，我們所需要思考的是能否修改本系統的設計，使得在較低容量的記憶體環境下，仍能夠處理資料量大過於記憶體容量的 Volume Data，例如自行設計一組 Page Swap 機制，利用硬碟空間當作暫存，並將需要計算的部分搬移至主記憶體。

另外，我們目前所設計出的系統，均為單機環境(Single Alone)設計。但因使用者介面與主程式已設計為兩獨立元件，在未來有新的需求時，能夠將主程式以另外不同的型式提供給使用者，例如以不同的視窗開發程式(MFC 或 BCB 等)實作，或者延伸至 Client/Server 架構以實現 Web Service 供使用者從網路上操作某個影像分析應用的服務，透過 Java Applet 等 Client 端程式作業而不需要在使用者的電腦上安裝過於昂貴的配備即可完成。

參考文獻

[1] **Computation of brain neurodegeneration in the Alzheimer's fly**
P. C. Lee, Y. R. Huang, H. M. Chang, A. S. Chiang, Y. T. Ching, Dept. of Computer Science, National Chiao Tung University

[2] **Toward designing an automatic method to count neuron cells in the fly brain**
Meng-Chih Chen, Yu-Tai Ching, Dept. of Computer Science, National Chiao Tung University

[3] **3D Objects Segmentation Based on the Balloon Model**
Shu-Wei Chou, Yu-Tai Ching, Dept. of Computer Science, National Chiao Tung University

[4] **A Geometric model for active contours**, V. Caselles, F. Catte, T. Coll and F. Dibos, Numer. Math. Vol. 66, pp. 1-31, 1993.

[5] **Gradient Vector Flow Models for Boundary Extraction in 2D Images**
Gilson A. Giraldi, Leandro S. Marturelli, Paulo S. Rodrigues
LNCC–National Laboratory for Scientific Computing

[6] **Active Contours, Deformable Models, and Gradient Vector Flow**
Chenyang Xu and Jerry L. Prince

[7] **Amira - http://www.amiravis.com/**
Advanced 3D Visualization and VolumeModeling Mercury Computer Systems

[8] **NIH Image -- http://rsb.info.nih.gov/nih-image/**
A public domain image processing and analysis program for the Macintosh
Scion Corporation

[9] **Design Patterns: Elements of Reusable Object-Oriented Software**
Erich Gamma, Richard Helm, Ralph Johnson, John Vissides
Addison Wesley Professional Computing Series

[10] **Design Patterns Wiki:**
http://en.wikipedia.org/wiki/Design_pattern_ (computer_science)