第四章

軟體設計

除了硬體的發展外，還需要軟體的配合，系統才能發揮適當的功用並增加額外的效能。關於兩邊連線部分，我們使用 WinSock 來建構網路通訊介面，並藉由 TCP/IP 通訊協定與 Ethernet 來達成兩端連接。另外，遠端操作系統除了遠端操作(teleoperation)，如能增加遠端呈現(telepresence)的效能，可增強系統的整體表現性，於是我們利用了 OpenGL 來建立互動式場景，以增加互動時的可見性與真實性。本章首先解釋在此系統中所使用的多執行緒技術，接著對 TCP/IP 網路通訊協定與 WinSock 以及 OpenGL 作簡單的介紹，最後我們提出雙邊雙迴圈的方法，並且利用本章先前幾節所提的相關技術，將系統的軟體部份整合起來，圖 4.1 表示本研究中所使用的平台(Windows NT)、程式語言(Visual C++)以及相關應用程式介面 (WinSock/OpenGL)。

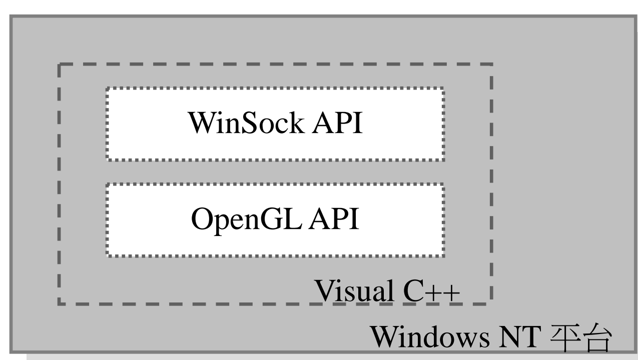


圖 4.1　軟體架構示意圖

4.1 多執行緒技術

在許多實際的應用上，有時候我們必須同時間去執行許多事，而不是等一件事做完後再去做另一件事，比如說，在本論文中的遠端操控系統，除了要提供操作者即時力回饋外，同時也要提供即時的視覺回饋，這些事必須同一時間都在做，而且很明顯地，不是先作其中一項再換另外一項，這時候就必須使用多執行緒(multithread)的技術。

多執行緒能夠同時提供多個執行緒(thread)的功能，通常執行緒所指的是在一個程式中的副程式，多重執行緒將一個程式分割成幾個小模組，每個模組都在一個特定的執行緒當中執行，這就宛如平行處理一般，但是卻利用了多工的能力，表面上使用者並未感覺到多重管線的運作方式，但是實際上程式執行的效率卻提高了。舉例來說，單執行緒就好像大型量販店只有一個結帳出口，如果顧客採買的東西只有一點點，那麼就可以很快速的結帳，但如果採買的東西很多，則結帳的時間就可能很久，而其它欲結帳的顧客就必須等待，而多執行緒就好像有很多的結帳出口，就算一個出口停頓了很久，也不會影響到其它的出口，如此可大幅的提升運作效益。圖爲一般程式設計也就是單執行緒的執行順序和時間關係，圖則爲多執行緒的執行順序和時間關係。此外，利用多執行緒的技術來設計程式，不但可提升效率，更可以簡化程式架構，我們將在本論文中，採用雙執行緒，分別處理力回饋與視覺呈現，至於詳細的流程架構，留待後面的章節再作完整的介紹。

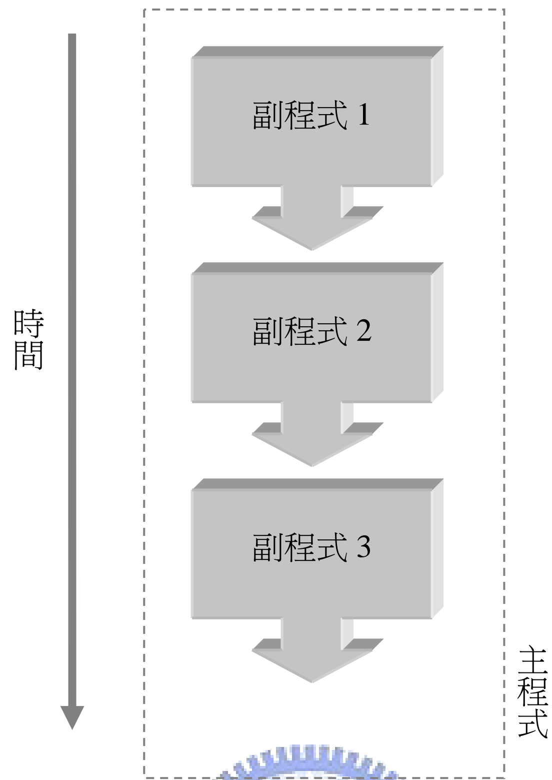


圖 4.2　單執行緒的執行順序和時間關係

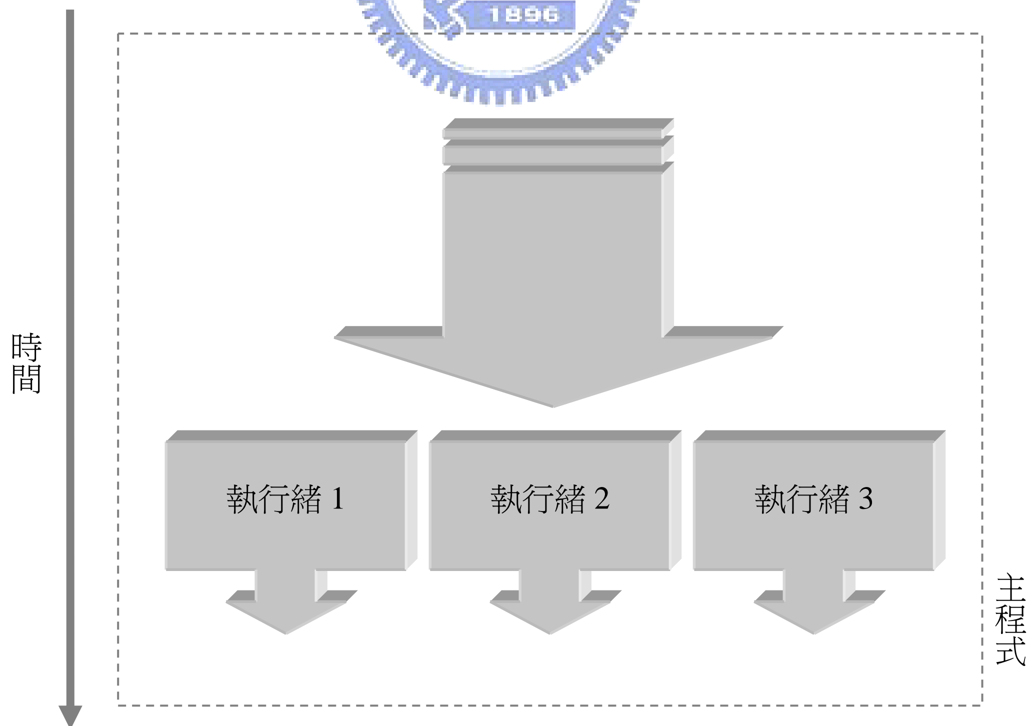


圖 4.3　多執行緒的執行順序和時間關係

4.2 TCP/IP 網路通訊協定

在本研究中我們使用了網際網路來當作我們的通訊媒介，並且利用 TCP/IP 通訊協定來作為傳輸資料的協定，為了之後能夠清楚解釋網路連線的狀況，於是在此先將其相關發展與通訊協定稍作介紹。

約在 1960 年代末至 1970 年代間，美國國防部所屬的一個研究部門 ARPA(Advanced Research Project Agency)為了實現異源網路(Heterogenous Network)彼此間能夠相互連接通訊，進行了一項 ARPANET 網路實驗計畫，而 ARPANET 網路即是目前網際網路的始祖，當時 ARPANET 網路上所使用的電腦間通訊協定稱為網際網路傳輸協定(IP，Internet Protocol)，此協定後來為了符合更高的傳輸品質要求而被修改為傳輸控制協定(TCP，Transmission Control Protocol)，後來 ARPA 要求所有連上 ARPANET 網路的電腦必須使用 TCP/IP 通訊協定，此後以 TCP/IP 為標準的網路越來越多。ARPA 為了推廣 TCP/IP 通訊協定，資助加州柏克萊大學將 TCP/IP 加入 BSD UNIX，隨著 80 年代 BSD UNIX 在全美各大學廣泛流行，TCP/IP 通訊協定也廣為大家所採用，為目前網路上最主要的傳輸協定。

國際標準組織 ISO (Internal Standards Organization)利用 OSI (Open Systems Interconnect)模型，將網路規範為七個層級，由低至高分別為實體層、網路鏈結層、網路層、傳輸層、會議層、表達層以及應用層，用來表示所有的網路，並且清楚定義每個不同層級的功能、標準與參考模型，如圖 4.4 所示：

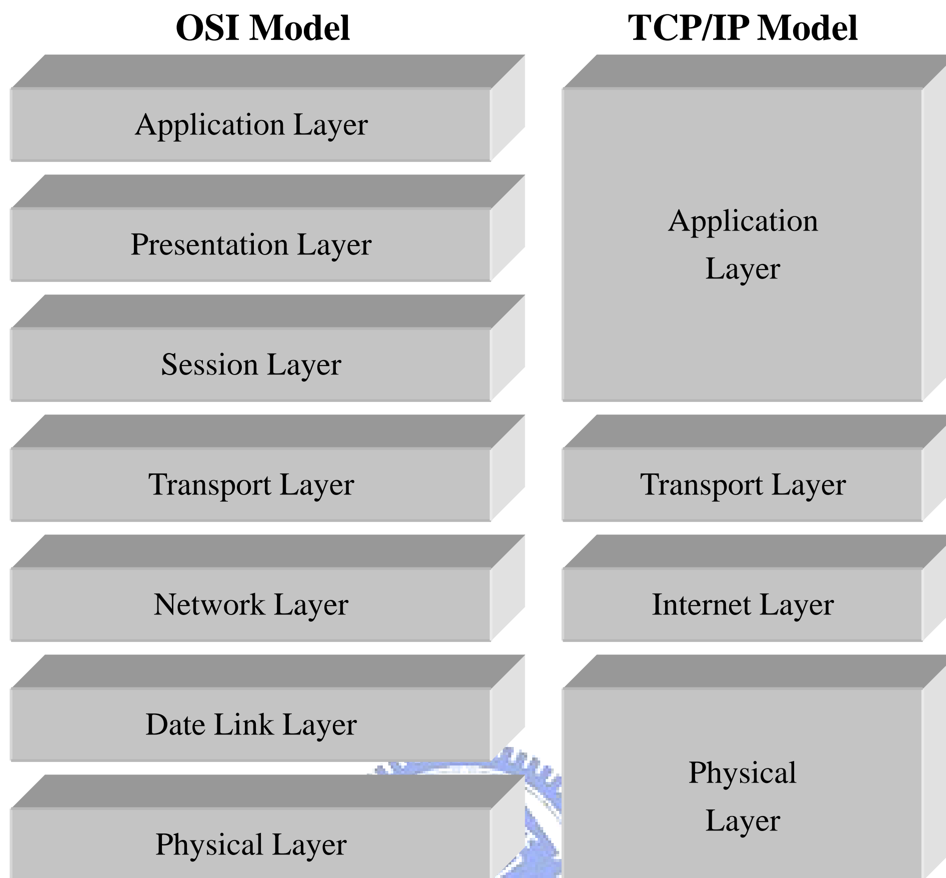


圖 4.4　國際標準組織 OSI 模型與 TCP/IP 模型

OSI 用七個層級的模型來代表一個實體網路(包跨軟體與硬體)，其最下層爲最低階的實體層，最上層則爲最高階的應用層，各層的功能如下：

第一層（實體層 Physical Layer）: 爲 OSI 的最低層，負責資料位元在實體傳輸媒體上的傳輸，主要的內容有資料實體呈現與傳輸的規格。

第二層（資料連結層 Data Link Layer）:負責確認實體層在連結時資料的正確性，包括有資料傳輸的錯誤偵測及錯誤更正。

第三層（網路層 Network Layer）;負責處理網路路徑的選擇、網路壅塞以及流量的控制。

第四層（傳輸層 Transport Layer）:負責兩端點間的資料傳輸控制服務，以及資料切割與重組等，讓接收端可正確無誤的依序收到，TCP 就是屬於傳

輸層的協定。

第五層 (會議層 Session Layer) :建立傳輸時所須遵守的規則，爲使用者於傳輸層間之介面，提供交談的建立、終止與管理並處理同步訊號。

第六層 (表現層 Presentation Layer) :負責將傳輸的資訊以有意義的形式表達給網路使用者，其中包括了字碼的轉換、字碼的編碼與解碼資料格式的轉換、資料壓縮與解壓縮。

第七層 (應用層 Application Layer) :爲 OSI 的最高層，位於網路應用的範疇，提供各式各樣的網路服務，如 WWW 服務、FTP 檔案傳收、Telnet 服務以及 E-MAIL 等

TCP/IP 通訊協定並不是只有 TCP/IP，實際上它是很多協定的組合，TCP/IP 協定家族是由 TCP、UDP、IP、ARP、RARP、ICMP 等所組成的，以下我們將針對 IP、TCP 與 UDP 作一介紹：

IP (Internet Protocol)/網際網路協定：

IP 位於網際網路層，主要工作是切割封包與選擇封包傳送的路徑，由於 IP 只負責將資料傳送到目的地，而不作任何錯誤檢查與控制，因此爲非可靠性傳輸，需要靠上層的 TCP 作偵錯的動作。

TCP(Transmission Control Protocol)/傳輸控制協定：

TCP 位於傳輸層，採用連接導向模式，具有資料接收確認回應與兩端相互維護封包順序號碼之特性，爲可靠性傳輸，適合用在需要可靠傳輸而不希望資料傳輸錯誤的地方。

UDP(User Datagram Protocol)/使用者資料傳施協定：

UDP 位於傳輸層，採用無連接模式，對於發送端送出的資料封包並不具有順序號碼，接收端也不會有所回應產生，雖然較不可靠，但與 TCP 比較起來具有較少的額外負擔(overhead)，此外，UDP 還有可作多點投射(multicasting)與廣播(broadcasting)的好處。

4.3 網路連結實作

WinSock 是 Windows Sockets 一詞的縮寫，由 Marting Hall 在 1991 年所提出的，並且在 1993 年，一個公開的視窗網路程式發展介面 Windows Sockets 1.1 版正式發行。WinSock 並不是指認何具有實體的程式或軟體，而是指一套公開在 Microsoft Windows 下發展網路程式的應用程式介面(Application Programming Interface，API)，一般而言，當我們再寫程式時需要依靠寫好的函式庫以及系統呼叫來達成我們的目的。視窗環境即藉應用程式介面提供了這樣的服務，其定義了若干個函式，包括每個函式的呼叫方式、回傳資料型態、變數個數及函式功能等，程式設計者按照這個規格，使用這些函式以取得系統服務。總之，Winsock 應用程式介面即是位於 TCP/IP 與應用程式間的介面，使得在視窗環境中的網路應用程式與 TCP/IP 傳輸協定核心間的溝通有了統一的規格，如圖 4.5 所示。

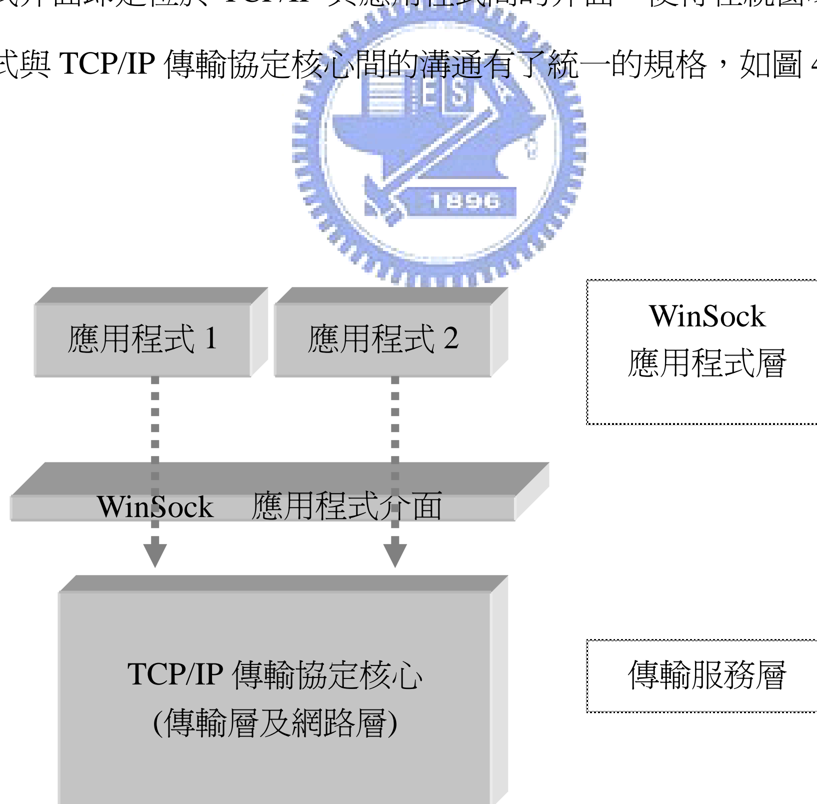
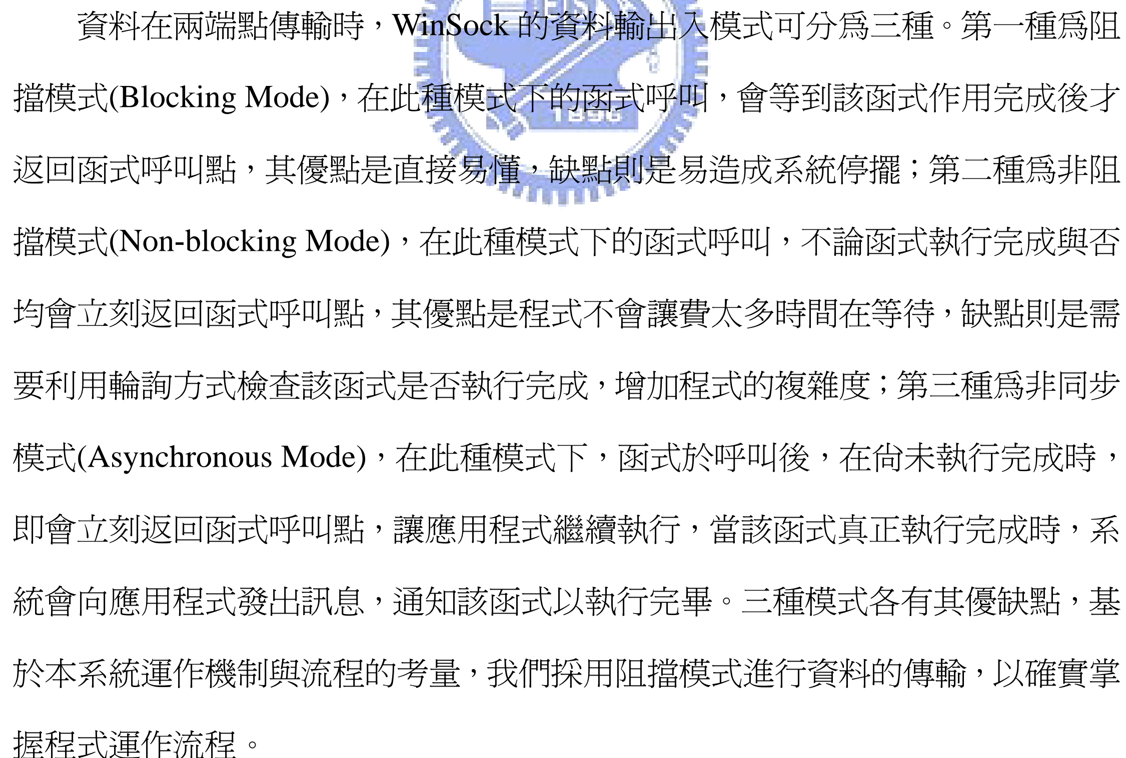


圖 4.5 WinSock 應用程式介面

在網路的實作上，我們採用無連接式(UDP)的 Server-Client 模式，如圖 4.6 所示，首先呼叫 WSAStartup()函式向作業系統要求使用 WinSock 的動態函式庫，接著在兩端各開啓一個 socket，socket 可抽象地想像成資料的傳輸口，利用這個傳輸口和遠端的機器互相傳送資料，並指定本地 IP 位址和 port number 給未定名的 socket，等一切設定都指定就緒後，Client 端透過 socket 將資料傳送給 Server 端， 待 Server 端接收完成後，將所收到的資料進行計算與處理，再透過 Server 端的 socket 將資料傳回給 Client 端，Client 端將接收到的資料再次進行計算與處理，如此即完成一次完整的傳送與接收迴圈，之後按照這樣的迴圈如此反覆下去，直到操作結束，而當要中斷兩端點的連線狀態時，Server 端與 Client 端均需利用 closesocket()函式關閉 socket，並且呼叫 WSACleanup()函式註銷使用 WinSock 的動態函式庫。



資料在兩端點傳輸時，WinSock 的資料輸出入模式可分爲三種。第一種爲阻擋模式(Blocking Mode)，在此種模式下的函式呼叫，會等到該函式作用完成後才返回函式呼叫點，其優點是直接易懂，缺點則是易造成系統停擺；第二種爲非阻擋模式(Non-blocking Mode)，在此種模式下的函式呼叫，不論函式執行完成與否均會立刻返回函式呼叫點，其優點是程式不會讓費太多時間在等待，缺點則是需要利用輪詢方式檢查該函式是否執行完成，增加程式的複雜度；第三種爲非同步模式(Asynchronous Mode)，在此種模式下，函式於呼叫後，在尚未執行完成時，即會立刻返回函式呼叫點，讓應用程式繼續執行，當該函式真正執行完成時，系統會向應用程式發出訊息，通知該函式以執行完畢。三種模式各有其優缺點，基於本系統運作機制與流程的考量，我們採用阻擋模式進行資料的傳輸，以確實掌握程式運作流程。

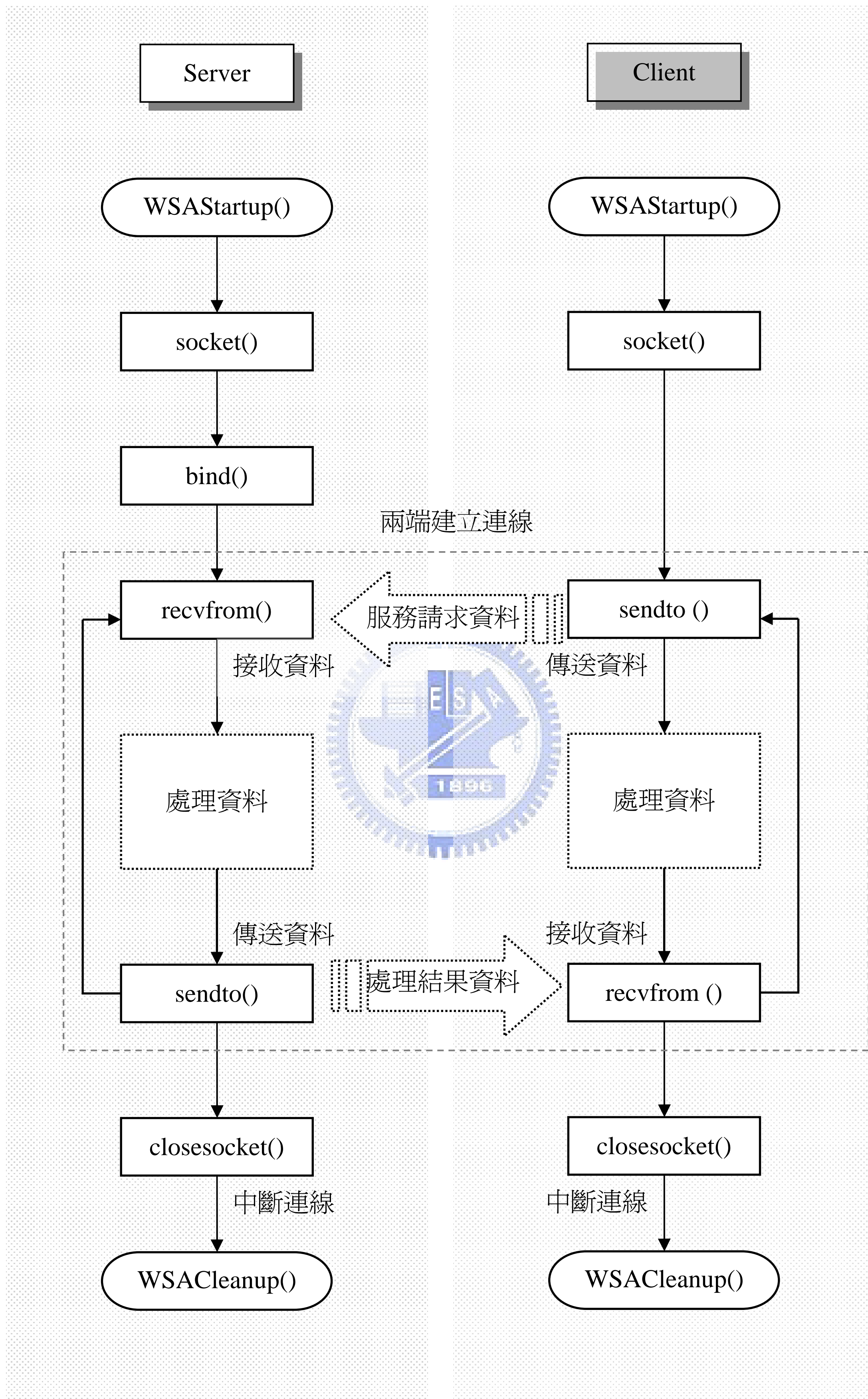


圖 4.6　無連接式(UDP)的 Client-Server 程式流程圖

4.4 虛擬場景實作

OpenGL 是在 1990 年代初期由 SGI 公司的 IRIX GL 標準所演化而來的，目前則當作 Unix、Linux、Mac OS 與 Windows 作業系統的 2D 與 3D 模型塑照，許多遊戲如 Quake 都用到這類技術。SGI 控制了 OpenGL API 的授權，而 OpenGL Architecture Review Board (ARB) 是一個獨立委員會，負責這個標準的維護與改進，並訂定詳細的計畫。OpenGL 1.0 版於 1992 年七月一日發行，最新的是 OpenGL 1.4 版。此外，爲了因應 OpenGL 在 Windows 遊戲上越來越常被使用，微軟另外開發了一套 DirectX，專供 Windows 平台遊戲使用。不過 DirectX 的遊戲並不容易移植至其他非 Windows 平台使用，所以 OpenGL 依然會是可攜性與開放標準的良好選擇。當然，市面上的繪圖軟體或函式庫體種類繁多，如 OpenGL、WorldToolKit 與 DirectX 等各有其優缺點，而 OpenGL 因具有可靠性、可攜性以及發展性，是一文件完備的應用程式設計介面，基於這些優點及跨平台的考量，我們採用 OpenGL 作爲本系統的繪圖函式庫。

OpenGL 並非程式語言，它是一種 API (Application Programming Interface，應用程式設計介面)，而此種繪圖 API 是程序式 (procedual) 而非描述式 (descriptive)，設計時不須描述場景，而必須規定出達到外觀某個特定效果的步驟，這些步驟就是對整套 API 的呼叫，其中包含了可攜性極高，超過 200 個指令與函數，這些指令都是用來畫出各種圖形元件，如空中的點、線、以及多邊形。此外，OpenGL 也支援燈光 (light) 與陰影 (shading)、材質貼圖 (texture mapping)、混合 (blending)、透明度 (transparency)、動畫以及其他許多特效。當應用程式發出 OpenGL API 函數呼叫時，指令會放到指令緩衝區中(command buffer)，接著座標點(vertex)資料會先被轉換並進行燈光運算，然後將資料提供給 pipeline 的 rasterization(掃描顯像)部分，rasterizer 將依據幾何位置，顏色與材質

資料建立影像，並且影像會被放到 frame buffer 中，frame buffer 是圖形顯像裝置的記憶體，最後將影像顯示到螢幕上，以上敘述如圖 4.7 所示。

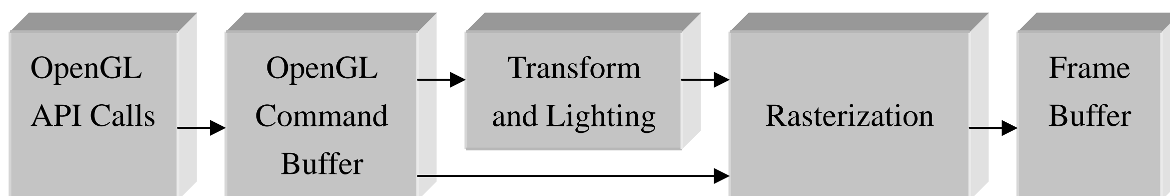


圖 4.7　OpenGL 呈像管線(Pipeline)

實作上，OpenGL 是一個標準的程式庫，我們使用 C 語言在 Windows NT 平台上進行撰寫，並且呼叫 OpenGL 的程式庫。在大多數的平台上，OpenGL 程式庫都伴隨有 OpenGL 工具程式庫(utility library)簡稱 GLU，檔名為 glu32.dll，此工具程式庫是一組專司經常性工作的函數，如特殊的矩陣運算，或支援各種常用的曲線或曲面。為了在 Microsoft 的工具中建立使用 OpenGL 的程式和呼叫 DLL 中的函數，必須匯入 opengl32.lib 以及 glu32.lib 這兩個程式庫，並且將程式與匯入程式庫(import library)相連結。另外，我們使用了 GLUT (OpenGL utility toolkit) 工具集程式庫，來增加設計時的方便性。圖 4.8 介紹了本研究中關於軟體繪圖程式部份，而在建構虛擬場景時我們利用了以下幾種函式，首先呼叫 glutInitdisplayMode()函式，告訴 GLUT 程式庫建立視窗時要採用何種模式，在此我們採用雙緩衝區視窗以及 RGBA 色彩模式；接著呼叫 glutCreateWindow()函式，用來在螢幕上建立場景視窗；glutReshapeFunc()函式，用來處理當視窗大小改變時新的設定值；glutTimerFunc()函式，用來建立繪圖場景的無窮迴圈，產生具有連續動畫的效果；glutDisplayFunc()函式，告訴 GLUT 當要繪圖時必須呼叫那個函式；最後呼叫 glutMainLoop()函式，用來啓動 GLUT 主要的處理迴圈，包含按鍵、滑鼠、計時器、繪圖與其他視窗訊息，直到程式結束時才會返回。

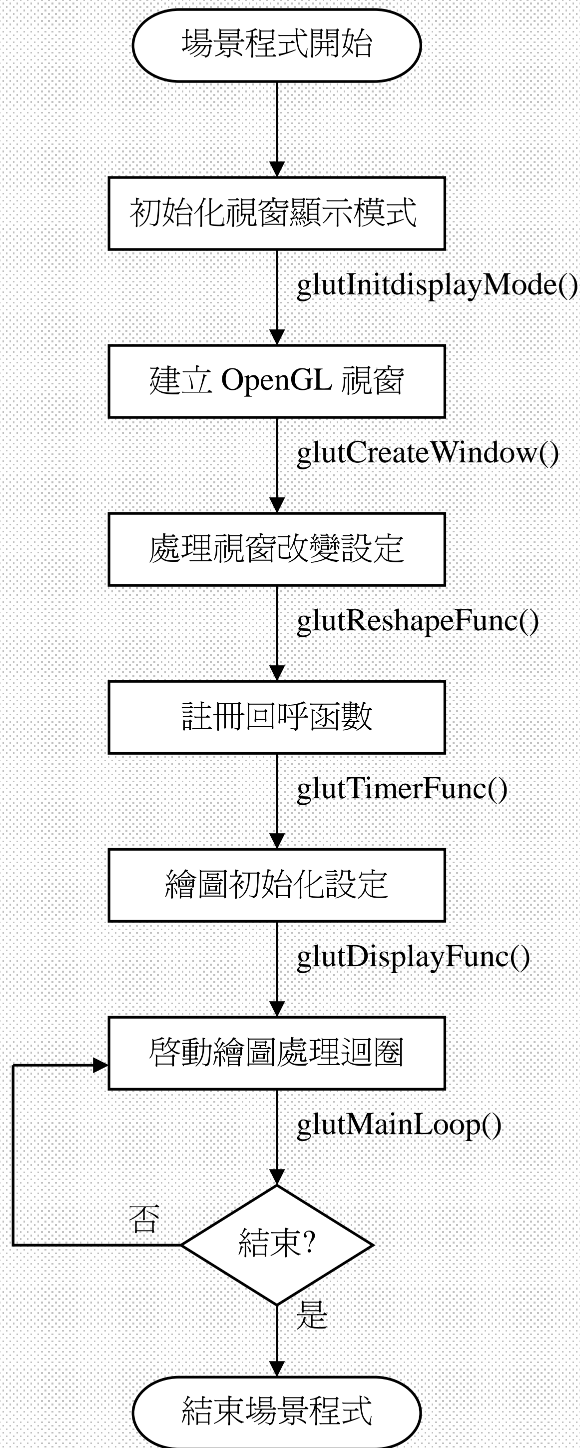


圖 4.8　　場景程式流程圖

4.5 雙邊雙迴圈整合

在之前的章節裡我們有提到，要讓人感到流暢、滿意的力更新頻率大概要500Hz，而影像更新頻率也要30Hz，典型的單迴圈程式無法同時滿足這些需求，於是我們採用雙執行緒的技術，分別處理力迴圈以及影像迴圈，如圖4.9所示。此外，將力迴圈與影像迴圈分開執行，除了可提高力更新頻率外，更可依據觸覺和視覺的不同需求而個別處理，如在強制性多工(Preemptive)的作業系統中(Windows NT / UNIX)，透過調整執行緒的優先權(priority)，可適當的分配系統資源。

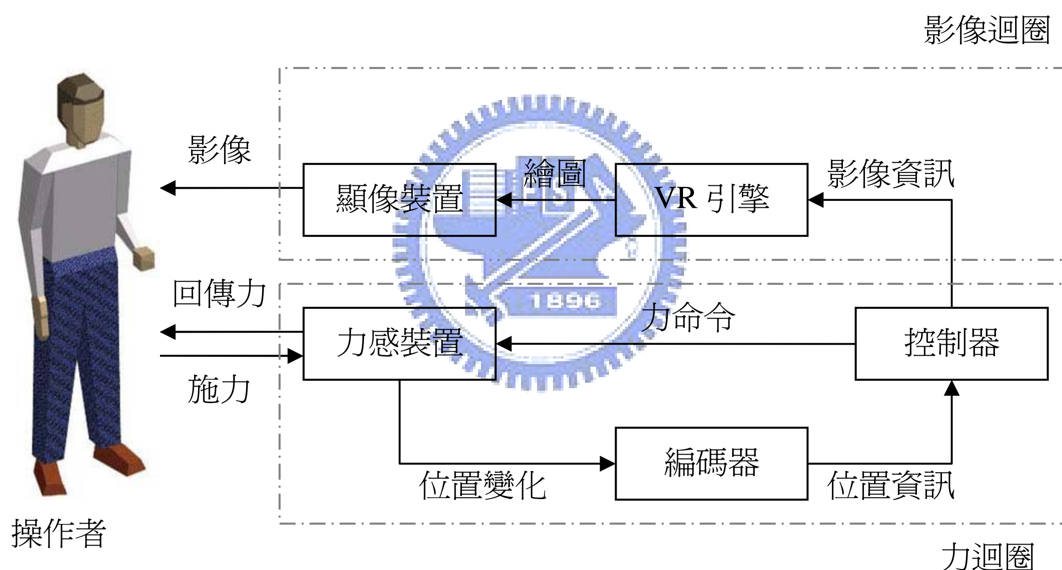


圖 4.9　系統雙迴圈流程示意圖

在雙執行緒的程式中，首先力迴圈接收來自遠端的力命令，藉由力回饋搖桿輸出遠端所施予的力量，接著近端操作者施予一力量抵抗，馬達上的編碼器讀取搖桿位置變化量，經由力感模型計算求出所應回傳的力大小，最後再將力命令傳回給遠端系統。而在影像迴圈中，由於執行緒具有資源共享的功能，我們可以經由全域變數(global variable)得知搖桿位置的變化量，再透過VR引擎畫出影像，最後由螢幕顯示出虛擬場景，以上詳細的流程如圖4.10所示。

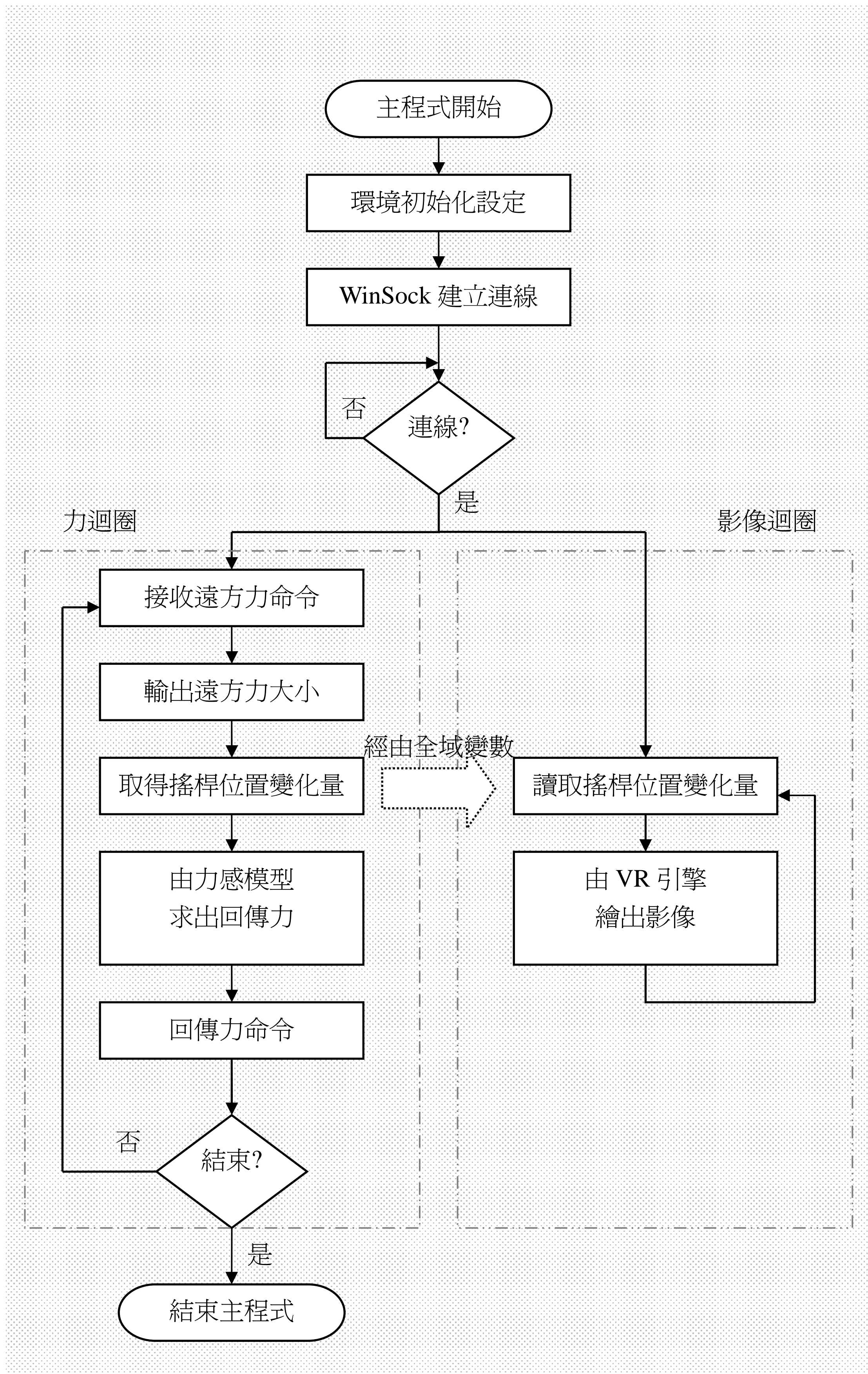


圖 4.10　系統雙迴圈程式流程圖