第二章

相關研究

2.1 運動計畫演算法

運動計畫演算法是要替環境中的物體，找出一條從指定的起始狀態到目標狀態的運動過程。這項研究一開始是起源於機器人學(Robotics)，主要的目的是要讓機器人可以不受到環境中障礙物的阻擾，順利的抵達目的地位置。之後經過不斷的研究發展，這些技術也被廣泛的應用到許多不同領域的問題上，包括產生 3D 虛擬實境中的角色動作動畫[19][26]、模擬現實中的人群[11][15]以及輔助遊戲中的操作介面[4][25]等。但不管應用領域有多少差別，其本上我們都還是可以把運動計畫的問題分成兩部份來討論，分別是碰撞偵測(Collision Detection)和路徑搜尋(Path Finding)。

碰撞偵測的目的是要檢查出被規劃物體的狀態是否合法。例如在[19]的研究裡，他們希望規劃出來的動作不會有碰撞發生，所以當角色在某個狀態下的動作會和場景發生碰撞時，角色的這個狀態便是不合法的。組態空間(configuration space)是過去一種常見的碰撞偵測解決方法，這種方法會事先把物體的所有可能狀態計算出來，並將它們是否合法的結果存在一張組態空間表裡。之後當程式執行時，它便可以直接用查表的方式來進行碰撞偵測。這種方式雖然有著執行速度快的優點，但因為必須事先列舉出物體的所有可能狀態，並用記憶體空間來儲存這些組態，所以通常都只會應用在自由度低且場景範圍小的運動計畫裡。街景圖(Roadmap)[27]是另一種常見的碰撞偵測解決方法，這種方法和組態空間的方法類似，差別則是在於它並沒有計算出整個組態空間，而是只取樣

(sample)出其中比較有可能被使用到的部份。之後當程式執行時，程式便會盡量利用這些已經被取樣到的狀態來進行運動計畫，並動態的將沒被取樣到的狀態更新到街景圖裡。這種方式因為不須要列舉出物體的所有可能狀態，所以可以減少許多事前的準備時間和必須被使用到的記憶體空間。但因為在執行時必須花費時間去更新它的組態空間表，所以執行速度相對的會比較慢。

路徑搜尋的目的是要從物體的所有可能運動組合裡，找出一條能符合目標要求的運動路徑。常見搜尋的方式有「規則導向 (rule-based)」、「目標導向 (goal-based)」和「學習演算法 (learning algorithm)」三種類型。規則導向的搜尋方式，是根據程式設計者預先設定好的搜尋順序來進行搜尋。這種方法有著容易實作的優點，但因為搜尋順序無法隨著環境而改變，因此有著缺乏彈性的缺點。目標導向的搜尋方式，是由程式設計者根據所希望的目標狀態來定義出一個比較狀態的演算法，藉由這個演算法來計算出搜尋目標的優先程度，藉此來動態決定搜尋的順序。這種方法因為能根據不同的環境狀態來改變它的搜尋順序，所以有著較高的彈性及搜尋效率，缺點則是必須花費比較多的時間去設計它的演算法。學習演算法的搜尋方式和目標導向的搜尋方式大部份都相同，不同的部份是它還會把程式執行的結果當成一種經驗儲存起來，並把這些經驗納入演算法的考慮範圍裡。這種方法的優點是能夠藉由累積經驗的方式來增加智慧，藉此補足演算法設計上的不足。缺點是當環境的狀態不停改變且改變的幅度都很大時，所累積的經驗可能會毫無用處，甚至導致更差的結果。

2.2 角色動畫的生成

在 3D 即時虛擬環境裡，角色動畫的生成方式基本上可以分成「以動力學模型為基礎 (dynamics model based)」和「以圖形為基礎 (graph-based)」兩種類型。以動力學模型為基礎的生成方式，是以動力學為依據，讓程式透過即時計算的方式來產生角色的動畫。例如根據關節(joint)之間的角度、關節所支撐的重量和整體的平衡感，來計算出角

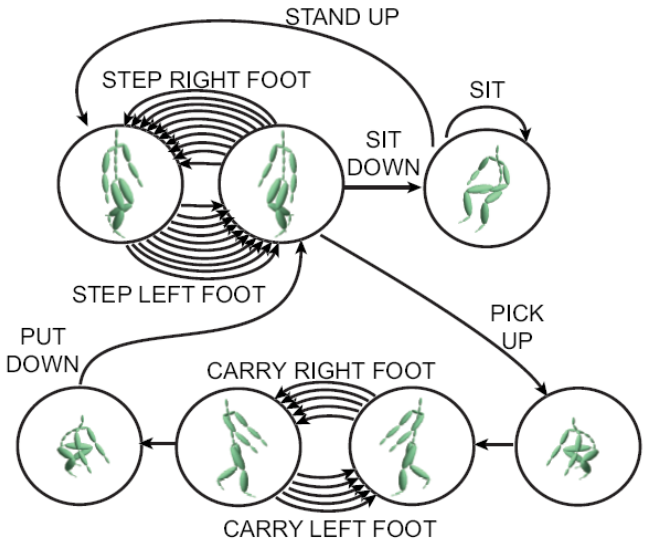

圖 2.1：動作圖的範例[29]

色走路[1][9][10]、跑步[12]或騎車[13]的動作。這種方法的優點是能產生出許多複雜且有彈性的動作，缺點則是須要比較大的計算量，且因為設計者通常很難把動力學模型設計的很完善，所以計算出來的結果往往會有著不真實或不自然的缺點。以圖形為基礎的生成方式，是由程式設計者事先定義好一些動作片段，並將它們之間的連接關係儲存成一個動作圖(Motion graph)[17]。之後當程式執行時，便直接根據這個動作圖來決定所要產生動作的順序，並用動作混合(motion blending)的方式來組成連續的動畫。圖 2.1是一個簡單的例子，圖中的節點代表著動作片段，程式可以用動作混合的方式來將兩個節點的動作片段混合成站起來(STAND UP)、坐下(SIT DOWN)或其它不同種類的連續動作。這種以圖形為基礎的動畫生成方式便是本研究裡所使用的方法，有著計算量小、可重覆利用和易於更新等優點，但也有動作單調和缺乏靈活性的問題。

動作單調和缺乏靈活性的原因，主要是因為動作圖的大小是有限的。一個出拳的動作在動作圖裡可能只能找到兩三種不同的選擇，程式不一定能從這些選擇裡找出符合使用者要求的動作，且當出拳的次數頻繁發生時，很容易就會被使用者發現是幾種相同的動作在被重覆使用。因此有研究提出以動態調整動作混合參數的方法將兩種動作進行混

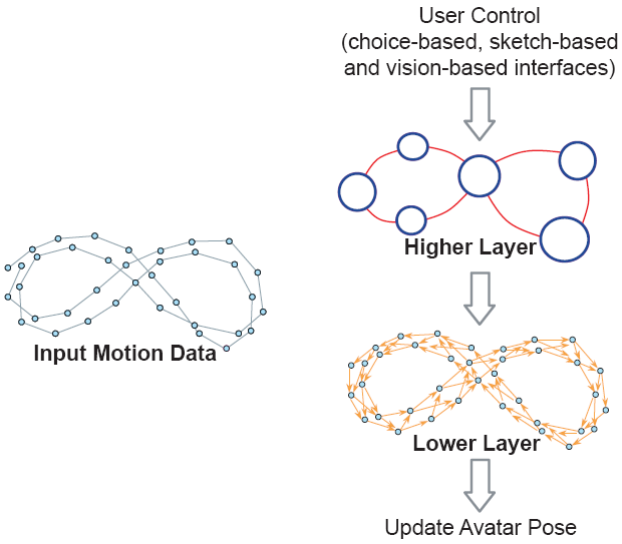

圖 2.2：雙層架構的動作圖[30]

合，藉此產生出許多不同的動作風格[7]。例如把走路和彎腰前進的動作進行混合，可以產生托著腳走路的動作，而根據混合參數的不同，又會使托著腳的動作有程度上的差別，藉此增加動作圖所能產生的不同連續動作。另一種方法則是將動作片段切細，藉此讓動作圖的節點及邊變多，使其達到動作增加的目的。但這種方法也會增加程式每次搜尋動作圖所須花費的時間，因此有許多研究便是在想辦法減少程式搜尋動作圖的時間。在[18]的研究裡，他們在動作圖裡增加了一種名為中心節點(hub nodes)的新類型節點，這種節點是動作圖裡比較重要的節點，會比一般的節點有著更多的邊。這種只在部份節點增加邊的方式，可以有效的增加動作圖的邊數，又不會使程式搜尋動作圖的時間增加太多。另外在[30]的研究裡，他們提出了一種雙層架構(two layer structure)的動作圖概念(圖 2.2)，這種架構會將動作圖分成高層(higher layer)和低層(lower layer)兩種類型。高層的動作圖是以動作片段的類型去組織動作圖，會有著比較少的節點及邊數。低層的動作圖則是相當於普通的動作圖，會有著比較多的節點及邊數。當程式在搜尋動作圖時，會先去搜尋高層的動作圖，再去搜尋低層的動作圖，藉此減少程式搜尋動作圖的時間。例如角色目前是一種走路類型的動作，當程式要搜尋角色的下個動作時，它會先去高層的

動作圖搜尋走路類型的動作可以轉換到那一種類型的動作，之後再到低層的動作圖裡去找有那些動作片段符合這種類型，並挑選出最適合的下一個動作。

但這兩種方法所能產生的變化性也還是有限，有些動作可能再怎麼組合也無法達成想要的結果。例如當動作圖裡的動作都是站著的姿勢時，電腦再怎麼調參數也都無法產生坐下的動作。所以有另一種研究方向，是偏重在直接修改這些動作資料的部份內容。藉由只調整少數動作參數的方式，讓修改後的動作除了能達到目標外，又能保持原本動作的真實性。例如讓演算法計算出手、腳和骨盆的空間限制(spatial constraints)，並用反向運動(Inverse Kinematics)的方程式來改變動作資料裡相關關節的位置資訊，產生搬箱子、丟球和走樓梯的動作[2][3][5]。這種方法相當於以動力學模型為基礎和以範例為基礎兩種方式的綜合。如果可調整的動作參數越多，動作可能就越不真實。相反的如果可調整的動作參數越少，動作的可變化性就越少。

2.3 反應機制的設計

角色反應機制的設計相當於在設計角色的智慧。「有限狀態機」(Finite-State Machine)是目前最常用來設計角色反應機制的方法，這種方法是把角色的狀態和狀態轉移條件都記錄在一個表格裡，之後當角色要和使用者進行互動時，便直接根據使用者的動作類型來觸發角色的狀態轉移。這種方法有速度快及容易實作的優點，但因為他的轉移條件只會改變自己的狀態，所以互動性比較差。「狀態-行為模型」(State-Action Model)[14]。是另一種用來設計反應機制的方法，這種方法除了角色的狀態和狀態轉移條件外，還記錄了狀態轉移條件所會連帶影響到的其它角色。例如角色 A 目前的狀態是「站著」，這個狀態有一個轉移條件是「看到石頭」，這個條件會連帶的影響到角色 B。那麼當角色 A 站著且看到石頭時，不管角色 B 有沒有看到石頭，角色 B 都要進行狀態的轉移。這種方法有著互動性佳的優點，但缺點是增加了設計上的困難。