前言

微軟首次發表 Kinect 當時，Matt Webb（倫敦設計顧問公司 Berg 的總裁）的真知灼見讓許多程式員、硬體駭客和玩家興奮不已：

『二次大戰與彈道為我們帶來了數位電腦。冷戰與分權為我們帶來了網際網路。恐怖主義與群眾監督為我們帶來了：Kinect。』

Kinect 的重要性何在

如同這些發明為 20 世紀造成了最根本的突破，Kinect 為科技帶來了革命。就像個人電腦（PC）或網際網路（Internet）的問世，Kinect 的出現，讓只供軍事和情報界享用、耗費數十億美元和數十年研究的成果，落入尋常百姓手中。

臉部辨識、步態分析（gait analysis）、骨架化、深度成像（depth imaging）—這些技術原本被應用在公共場所來偵測恐怖分子，現在突然可以應用在具創意的非軍事用途上：為軟體建構手勢介面、為個性化製造（personalized fabrication）建構廉價的 3D 掃描器、為簡單的 3D 角色動畫使用動作捕捉技術、利用生物辨識技術為失能人士建立定製化科技輔具…等等。

雖然 Kinect 的發展看來似乎廣泛而多樣，但可以簡單歸納為：這是史上第一次，電腦可以看得見。雖然過去這數十年來電腦已經能夠處理影像和視訊，但僅侷限在紅、綠、藍像素之上，無法提供人類視覺系統應當具備的能力：看到立體影像的能力、在空間中從事區分物體的能力、隨著時間和空間的推移追蹤人體的能力、辨識身體語言的能力…等等。Kinect 為攝影機及影像處理技術所帶來的革命，讓我們得以著手建構具備以上這些能力的應用程式，並以此為起點。Kinect 的到來讓即使是週末的玩家還是業餘的駭客也能夠創建出具備這些能力的應用程式。

如同之前個人電腦和網際網路所帶來的科技革命，視覺革命（Vision Revolution）也必定將導致具創意及創造力之應用比比皆是的驚人盛況。拿 Kinect 的到來與個人電腦及網際網路相比，聽起來可能有些荒謬。但請記住，當個人電腦首次被發明時，它只是玩家和愛好者的古怪玩具。網際網路則是因為政府的研究人員需要存取其他單位的主機而誕生的。這些技術只有在漸漸被人們拿來製作具創意及新穎的應用，並在最終成為我們日常生活中的設備後，才會在當代生活中扮演重要的角色。現在就拿 Kinect 來與個人電腦和網際網路相提並論，似乎很荒謬，但從現在開始幾十年後，當我們回頭看 Kinect 的時候，可能會拿它跟 Altair 或 ARPAnet 相比，把它當作只不過是嬰兒邁向新世界的第一步。

本書的目的在提供你所需要的背景知識和技能，讓你得以正確地建構出，能夠揭示這個新世界的應用。這些技能包括：

- 處理來自 3D 攝影機的深度資訊
- 分析及操作點雲（point clouds）
- 追蹤人體關節的移動
- 背景移除及場景分析
- 姿勢及手勢的偵測

本書前三章就會把這些技能全都傳授給你。你將學到，在 Processing 的程式設計環境中，實作這些技術的方法。我們會從最基本的技能（讀取 Kinect 的資料）開始，循序漸進地讓你建立起撰寫更複雜程式的能力。學習這些技能，不僅能讓你掌握相關的軟體程式庫或 API，還能讓你瞭解它們背後的運作原理，這讓你得以善加應用它們，即使技術的具體細節演變得非常快。

然而，即使能夠掌握這些基本技能，仍不足以讓你建構出充分利用此視覺革命的應用。因此，你還需要知道，廉價而容易取得的深度資料及骨架資訊將會對哪些領域造成徹底的改變。最後，本書將會對 3D 掃描、數位製造、機器人視覺以及輔助技術等領域提供概念性的介紹。你可以把這個部分的內容當成是在教你，取得深度和骨架資訊之後，可以把它拿來做什麼。這個部分將會談到如下的議題：

- 建構網格
- 為製造預備 3D 模型
- 定義及偵測手勢
- 顯示及操作 3D 模型
- 為行動範圍有限的人設計定製化輸入裝置
- 順向及逆向運動學

探討這些議題時，我們的焦點將會從簡單的 Kinect 資料處理向外擴展至一整個軟體工具集和技術。本書最後三章將會透過一系列深入的實驗來探索這些議題。我們將會撰寫一支程式，透過把 Kinect 當成掃描器，在 3D 印刷機上製作出實際的物體；我們將會製作一個遊戲，以協助中風病人進行物理治療；我們將會建構一支機器人手臂，以複製你的手臂動作。在這些實驗中，我們會先介紹相應領域背後的基本原理，接著會檢視我們新發現的程式設計知識與 Kinect 如何能夠將這些原理轉化成實際的行動。除了 Processing 與 Kinect，我們還會介紹實作每一種應用所必須用到的任何工具，包括 3D 建模程式以及微控制器。

本書並不會對這些議題做全面的介紹；這些議題所觸及的都是巨大、廣泛、錯綜複雜且引人入勝的領域。本書的目的在對這些領域做拋磚引玉的介紹—讓你具備足夠的背景知識和技術，因而能夠開始使用 Kinect 來製作有趣的專題，並期盼你的進展能夠激勵你按照本書所提供的線索做更深入的研究。

本書的訴求對象

本書的訴求對象是，想要瞭解更多關於如何使用 Kinect 創作互動應用的任何人，包括想要建構手勢介面之互動及遊戲的設計者；想要使用 3D 掃描器的製造者；以及想要著手使用電腦視覺科技的藝術家。

也就是說，若你是下列之一，本書將發揮最大的功效：一個新手程式員，想要學習更複雜的圖形和互動技術，特別是在三維空間的做法；或是一個高階程式員，想要快速學習 Kinect 的使用細節，以及瞭解能夠應用 Kinect 的領域。

以本書為起點，你不必是一個專業的圖形程式員，或是老練的 Processing 使用者，但如果你之前從未寫過程式，或許你可以從其他比較好的起點開始。

因此，我將會假定你對 Processing 這個創意撰碼（creative coding）語言已經有所涉獵（或是當需要用到 Processing 時，你有辦法自學）。你可以從 Casey Reas 和 Ben Fry 所著的《Processing 入門：互動式圖形實作介紹》（*http://books.gotop.com.tw/bookdetails.aspx?Types=&bn=A317*）、Dan Shiffman 所著的《Learning Processing》（*http://learningprocessing.com*）或之類的書籍學到你應該知道的基礎知識。本書的寫法是從介紹性的議題慢慢進入較複雜的程式碼和概念，所以在教你 Kinect 的時候，也會順帶介紹建立互動式圖形應用程式的基礎知識。開始的時候，我將會詳細講解每個例子，結束每個例子的時候，我會留下越來越多你需要弄清楚的細節。目標是將你從一個新手提升為一個自信的互動圖形程式員。

本書的結構

本書的目標是讓你有能力使用 Kinect 來建構互動式應用程式。也就是說，讓你成為，我在〈前言〉開頭所說的「視覺革命」的正式成員。成為這場革命的成員有許多好處。一旦你做到了，你將能夠讓一個看不見的鼓發出真實的聲音；印刷出物體的 3D 掃描副本；教機器人複製你的手臂動作。

然而，成為這場革命的成員，是要付出代價的。要加入這場革命的行列，你需要學習一系列基本的程式設計概念及技術。這些技能都是讓你獲得更進一步好處的根基，沒有這些基礎，你是無法擁有這些很酷的能力的。本書的做法是讓你一次學會一個技能，先從最簡單及最基礎的開始，再朝更複雜及更先進的邁進。我們會從簡單的像素開始，然後一步一步提升到錯綜複雜的三維手勢。

因此，本書的前半部可做為這些程式設計技能入門之用。在我們能夠進行機器人控制或對我們的臉進行 3D 印刷之前，我們必須從基礎開始。本書頭四章會說明，撰寫 Processing 程式使用 Kinect 送來的資料時，需要具備的基礎知識。

Processing 是一個創意撰碼環境，它採用了 Java 程式語言的語法，讓初學者輕易就能撰寫出包含圖形及其他媒體形式的互動程式。如前所述，本書會假定你具備 Processing（或同類型程式語言）的基本知識，但當你閱讀本書頭四章的內容時，我將會逐步為你建立及處理與 Kinect 最相關之更進階的 Processing 概念。這些概念包括：以迴圈處理像素陣列、基本的 3D 繪圖和定位，以及一些簡單的幾何計算。

我會盡量清楚、深入地說明每個概念。因為我並不想只是提供幾份「食譜」讓你照本宣科，而是希望你真的瞭解「基本食材」的味道，並因而能夠創造出你自己的「菜」，以及修改我在此處所提供的「菜」。有時，你可能會覺得，我把一些議題講得太詳細了，讓你無法承受，請堅持下去 — 當你試圖把你的想法實作出來時，你常會發現這些細節變得極其重要。

這是因為，除了 Kinect 的使用，你還需要知道許多相關的基本技能。如果你在使用 Kinect 的過程中能夠掌握它們，你也可以在使用 Processing 進行其他工作時將它們應用出來，這將可在你的工作中開啟許多新的可能性，而且最後真的能將你從初學者的程度提升起來。

要使用 Kinect 建構出吸引人的應用，需要用到三種基本技術：處理深度影像、轉換成 3D 以及存取骨架資料。從 3D 掃描到機器人視覺，所有這些應用都需要使用深度影像來量測物體的距離、將影像重構成三維場景，以及追蹤使用者身體各個部位的動作。本書的前半部會被用來介紹這些技術。我將會說明 Kinect 所提供的資料如何讓這些技

術成為可能、示範如何以程式碼來實作這些技術，並舉幾個簡單的例子，讓你看看它們可能有哪些用處。

深度攝影機

首先，你需要學會如何使用 Kinect 所提供的深度資料。Kinect 使用了一個紅外線投影器（IR projector）及攝影機來產生它前面之場景的「深度影像」（depth image）。傳統影像中每個像素所記錄的是光線的顏色，此光線是由場景在該部分反射至攝影機的；深度影像中每個像素所記錄的是來自 Kinect 的場景在該部分之物體的距離。當我們檢視這些深度影像時，它們看起來好像是古怪的黑白照片。它們看起來很奇怪是因為影像中每個部分的顏色所代表的並非物體的亮度，而是它的距離有多遠。影像中最亮的部分是距離最近的地方，而最暗的部分則是距離最遠的地方。如果撰寫一支 Processing 程式來檢視此深度影像中每個像素的亮度，我們可以推算出 Kinect 前方每個物體的距離。相同的技術再加上一點點巧妙的程式碼，我們還可以在距離最近的點移動時跟著它，並藉此替簡單的互動性追蹤使用者。

點雲

第一種做法是把深度資料視為只有二維。也就是把 Kinect 所捕獲的深度資訊當成平面影像來看，但實際上它所描述的是一個三維場景。本書第 3 章，我們會開始介紹如何把這些二維像素轉換成三維空間中的點。就深度影像中的每個像素而言，我們可以用 x-y 座標來看它在影像中的位置。也就是說，如果我們所檢視的像素距左上角 50 個像素、向下 100 個像素，那麼它的 x 座標為 50、y 座標為 100。但是像素還具有灰階值。而且從我們前面對深度影像的討論可以知道，每個像素的灰階值與影像的深度相對應。因此，該值代表像素的 z 座標。

一旦我們將所有的二維灰階像素全都轉換成三維空間中的點，我們所擁有的資料稱為點雲（point cloud）—也就是說，有一堆分離的點彼此靠近著漂浮在三維空間中，而且與 Kinect 前面之物體和人體相對應。你可以把點雲視為與像素化影像（pixelated image）等效的 3D 影像。雖然點雲從遠距離來看並無空隙，但如果我們湊近一點看，影像將會變成一堆分開的點，點與點之間可以看到空隙。若我們想將這些點轉換成平穩連續的表面，我們將需要找出用大量的多邊形連接它們以填補空隙的方法。這個過程稱為建立網格（constructing a mesh），而這也是本書在之後的實體製造和動畫章節將會進一步探討的議題。

不過，光是點雲本身就有許多我們可以著力之處。首先，點雲就是酷。你可以擁有你自己的 3D 現場演示，而且你的周圍就在螢幕上，

你可以從不同的角度來操作和檢視它，感覺上有點像活在未來。使用 Kinect，這是第一次，你所獲得的世界觀與你一般透過傳統攝影機所看到的完全不一樣。

你必須具備在 3D 空間中進行導航和繪圖的一些撰碼基礎，才有辦法建立出這樣的新觀點。當你要進行 3D 空間的操作時，我將會試著協助你避免一些常見的陷阱。例如，當你要進行 3D 空間的導航時，很容易迷失方向，使得你所描繪的形狀到最後看不見了。我將會說明 Processing 的 3D 座標系統，而且會介紹一些工具，讓你在其中進行導航和繪圖時不會被搞糊塗。進行 3D 繪圖時，另一個常叫人困惑的地方就是攝影機的概念。為了將來自 Kinect 的 3D 點雲轉譯成我們可以實際描繪在電腦螢幕上的 2D 影像，Processing 使用了一台虛擬的攝影機。當我們在 3D 空間安排好我們的點雲之後，我們會在該空間中一個特定的地點安置一台虛擬的攝影機，指向我們所描繪的點雲，主要是要拍一張照片。正如一台真正的攝影機會把它前面的物體扁平化成 2D 的影像，虛擬攝影機會對我們的 3D 幾何圖形做相同的事。攝影機所看到的一切，會按照它所看的角度和方式，被描繪在螢幕上。在攝影機視線以外的任何一切都不會被呈現出來。我將會告訴你如何控制攝影機的位置，好讓你想要看的（由 Kinect 送來的）所有 3D 雲點最後都會被呈現在螢幕上。我還會示範如何移動攝影機，好讓我們不必實際移動 Kinect 就能夠從不同的角度來檢視我們的點雲。

骨架資料

就某些方面而言，第三種技術不僅最簡單而且最強大。除了目前為止我們一直在使用的原始深度資訊，在一些額外軟體的協助之下，Kinect 還可以辨識人體，並且告訴我們，他們位於空間中何處。特別是，我們的 Processing 程式碼可以存取人體的每個部分在 3D 空間中的位置：我們可以得到手部、頭部、肘部、腳部…等等的確切位置。

相較於傳統的彩色影像，深度影像的一大優點是，電腦視覺演算法較容易處理。Microsoft 之所以要開發深度攝影機做為 Xbox 的控制器，並不是打算讓玩遊戲的人看看點雲有多酷，而是想讓 Xbox 上所執行的軟體能夠定位出人體，並找出人體各部分的位置。這個過程就是所謂的骨架化（skeletonization），因為軟體會從深度影像中的資料推斷出使用者的骨架（特別是，他的關節以及連接關節的骨骼）的位置。

只要使用正確的 Processing 程式碼，我們不必實作極其複雜的骨架化演算法，就可以取得使用者的位置資料。我們只要查詢我們感興趣之任何關節的 3D 位置，接著使用所取得的資料來建立我們的互動應用就行了。第 4 章中，我將會示範如何從 Kinect 的 Processing 程式庫

取得骨架資料，以及如何用這些資料來建立我們的互動應用。要建立真正精彩的互動應用，我們需要學會一些更複雜的 3D 程式設計技術。第 3 章中，使用點雲的時候，我們會探討到 3D 繪圖及導航的基礎知識。接著，我們會透過學習更進階的工具來添加一些技能，像是比較 3D 空間中的這些點、追蹤它們的活動，甚至加以記錄以備之後播放。為了在我們的程式中使用一些令人興奮的介面，我們將需要以這些新技術做為基本的語彙，好讓使用者能夠透過擺姿勢、做舞蹈動作和表演活動（以及許多其他自然的人類動作）來跟我們溝通。

一旦說明過使用 Kinect 的這三種基本技術之後，我們就可以繼續前進到本書開頭所提到可能吸引著你的一些酷炫的應用。本書的前提是，Kinect 真正令人興奮的地方，在於它打開了以往只有實驗室中使用昂貴實驗設備的研究員才能夠進入的電腦互動領域。有了 Kinect，諸如 3D 掃描及先進的機器人視覺技術，突然落入了使用 Kinect 以及對本書所提到的基礎知識有瞭解的任何人手中。但是為了能夠善加利用這些新的可能性，你需要具備一點實際應用領域的背景知識。要建立模仿人類動作的機器人，只知道如何存取 Kinect 的骨架資料是不夠的，你還需要對逆向運動學（研究如何安置機器人的各個關節以擺出一個特定的姿勢）有些熟悉。要建立可供製造或電腦圖形使用的 3D 掃描，只知道如何使用來自 Kinect 的點雲是不夠的，你還需要瞭解如何從這些點建構出網格，以及如何為 MakerBot、CNC 機械或 3D 印刷機上的數位製造預備及處理這些點。

最後兩章所要介紹的就是這些議題：數位製造的 3D 掃描以及機器人的 3D 視覺。

數位製造的 3D 掃描

第 5 章中，我們的焦點將從人體轉移到物體。我們會使用 Kinect 做為 3D 掃描器，以數位的形式來捕捉實物的幾何圖形，然後我們會為 3D 印刷機上的數位製造預備資料。我們會學到如何處理來自 Kinect 的點雲，以便把它們轉換成連續的表面或是網格。接著我們會學到如何以標準的檔案格式匯出此網格，這樣我們就可以在 Processing 之外來使用它。而且我會介紹幾個免費的程式，這些程式將能夠協助你整理網格，以及在製造之前對網格進行預處理。一旦我們的網格已經準備就緒，接著我們會研究，在一系列不同的快速原型系統上，該如何把它印刷出來。我們會使用 MakerBot 以塑料將它印刷出來，而且我們會把它遞交至 Shapeways，這個網站將以各種材質（包括砂岩以及鋼）印刷出我們的物體。

機器人的電腦視覺

第 6 章中，我們將看到 Kinect 的機器人應用。機器人視覺是一個超過 50 年的大議題。已有的成果包括月球上的自行機器人，以及裝配汽車的機器人。我們的機器人的腦袋將是一個 Arduino 控制器。Arduino 是 Processing 的姊妹專案，應用於電子裝置；透過 Arduino，我們輕易就能建立互動式電子裝置，正如 Processing 對互動式圖形應用所做的。Arduino 將聽取 Processing 送來的命令，並且控制機器人的馬達，以便執行這些命令。

我們將以兩種不同的做法來進行這一章的實驗。首先我們會根據 Kinect 偵測的結果，重現你的關節的角度。此做法屬於順向運動學（forward kinematics）；也就是，機器人最終的位置是把它的關節設定成一系列已知角度所獲得的結果。然後我們將重新設計機器人的程式，讓它能夠跟隨你的任何關節移動。這將是逆向運動學（inverse kinematics）中的一個實驗。此實驗中，我們不知道要如何移動機器人，我們只知道它的最終位置。我們必須教它如何計算每個需要變更的角度，以便獲得該結果。相較於順向運動學，這是一個更難的問題。一個正式的解決方案將涉及複雜的數學運算以及令人困惑的程式碼。我們的解決方案非常簡單並不複雜，但我們將對你在更進階的機器人應用中可能遇到的問題，提供有趣的介紹。

這幾章都不會對相關的領域做全盤的介紹，只會提供足夠的背景知識，讓你得以著手應用這些 Kinect 基礎知識來實現你的想法。

不同於頭四章試圖循序漸進地深入探討基礎技術的做法，最後這三章主要想讓你體驗一下 Kinect 應用的廣度及多樣性。後面這幾章中，我們不會循序漸進地對工作原理做全面的解釋，而會把它們安排成各自獨立的專題。每個專題皆發想自一個主題領域，而且會從頭到尾執行完畢。進行這些專題的過程中，我們將經常發現，我們需要做的事情不僅僅是撰寫 Processing 程式碼。我們還要訪談職業治療師、與使用輔助技術的病人合作、整理 3D 網格、使用 3D 動畫程式、焊接電路以及為 Arduino 寫程式。一路上，你將短暫接觸大量的新點子及工具，但無法像頭四章那樣獲得深入的瞭解。我們將迅速採取行動，而這將會令人感到振奮。你將不會相信，這些東西是你做出來的。

進行這些專題的每一個步驟，都將仰賴你在本書前半部所獲得的知識。所以當我們在探討這些基礎知識的時候，請全神貫注，它們是本書每個專題的基本構件，能夠充分掌握它們，將會比較容易打造出你想要建構的東西。

本書最後一章中，我們的視野將會被擴大。在你對 3D 程式設計以及 Kinect 的應用有所瞭解之後，我將告訴你下一步該怎麼做，好讓

你的應用能夠有進一步的發展。我們將探討 Processing 之外可以使用 Kinect 的其他環境及程式語言。這包括了以其他語言（例如 C++）所寫成的、具創意的程式庫，以及具互動性的圖形環境，例如 Max/MSP、Pure Data 和 Quartz Composer。此外，微軟自己也提供了一套開發工具，讓你得以把 Kinect 與 Windows 密切地整合在一起。我將說明每一個開發環境的優勢和機會，讓你感覺一下，你為什麼會想要去嘗試它。此外，我將告訴你，在每個領域剛起步的時候，你會用到的其他資源。

除了探索其他的程式設計環境，通常你還可以透過 3D 圖學的學習來對你的 Kinect 做進一步的使用。事實上，Processing 的 3D 繪圖程式碼係基於 OpenGL（這是一個被廣泛使用的電腦圖形標準）。OpenGL 是一個龐大、複雜、強大的系統，但 Processing 只使用了其中一丁點的功能。多學習一些 OpenGL 的知識，將可為你的 Kinect 應用打開各種更進階的可能性。我將會告訴你 Processing 之內和之外的各種資源，讓你能夠繼續你的圖學教育，以及建立出更漂亮、更引人入勝的 3D 圖形。

致謝

通常作者的致謝詞都會說，一本書的出版不是一個人能夠辦到的，而是集合眾人之力一起努力的成果。我自己也不例外。特別是，我所擁有的必要知識及寫作本書的能力，係直接得益於紐約大學（NYU）互動電子通訊研究所（ITP）各個老師卓越及耐心的指導。這本書沒有他們的協助是不可能完成的。

Dan Shiffman 的熱情激起了我對 Kinect 最初的興趣；他是一個教授、技術編輯以及朋友，他不倦的協助，讓我得以完成 Kinect 的學習以及撰寫相關的書籍；身為一個老師及作家，他所展現的能力，是我渴望達成的目標。

Kyle McDonald 和 Zach Lieberman 在 2011 年春季所開的一個為期七週的短期課程改變了我的生命。該課程讓我學到了我試圖在本書傳遞給讀者的許多技術和概念。希望我對這些內容的說明能夠達到他們傳授給我的一半。此外，Zach 希望本書加上藝術家訪談的建議，最後成為本書中我最喜歡的一個部分。而且在 Kyle 非常寶貴的協助之下，讓我得以用簡單的語言來表達他在數位製造之 3D 掃描的成果，這構成了第 5 章的主要內容。

感謝 Dan O'Sullivan（ITP 的系主任）和 Red Burns（ITP 的創辦人及守護神）為這個驚人的計畫提供空間和機構性的支援，並且建立一個環境，讓我有信心完成此計畫。

Lily Szajnberg 是我的第一個學生以及理想的讀者。本書之所以能夠提供最佳的解釋，都要感謝她求知若渴迫切的心，以及當我詞不達意時誠實以告。

我想要感謝 O'Reilly 的 Andrew Odewahn 和 Brian Jepson。Andrew 是第一個相信— 甚至在我之前 — 我能夠寫出一本書的人。此計畫在他初期回應的協助之下，從冗長的部落格貼文轉換成一本書。在 Brian 不斷及持續的努力之下，讓本書的內容無懈可擊。

感謝 Max Rheiner 創造本書所使用的 SimpleOpenNI 程式庫，以及充當本書的技術編輯，確保本書的正確性。沒有他的努力，本書的出版將會更困難而且很糟糕。

我還想要感謝接受訪談的所有藝術家：Robert Hodgin、Elliot Woods、blablablLAB、Nicolas Burrus、Oliver Kreylos、Alejandro Crawford、Kyle McDonald（再一次）、Josh Blake 以及 Adafruit 的 Phil Torrone 和 Limor Fried。你們的成果，以及希望看到更多類似成果的期盼，是我撰寫本書的動力。

我要十二萬分地感謝 MakerBot 公司的 Liz Arum 和 Matt Griffin 以及 Catarina Mota 協助我加快腳步塑造出好的作品，還有 Shapeways 公司的 Duann Scott 能夠確保我的作品及時送達。

最後要感謝我的家人和朋友以及 ITP 的同事在我寫書期間對我的容忍：我愛你們。

使用範例程式

本書的目的在協助你完成工作。一般而言，你可以在自己的程式與文件裡，使用本書的程式碼。除非重新製作並散布大部分程式碼，否則不需要與我們連繫以取得授權。例如，開發程式時使用書中的一些程式碼，並不需要取得授權。然而販賣或散布歐萊禮書籍的範例程式光碟，就需要取得授權。為了回答問題，引用這本書的內容或程式碼，並不需要取得授權。把書中大量的範例放到你自己的產品文件中，就需要取得授權。

我們會很感謝你在使用範例程式碼時註明出處，但這並非必要。出處說明通常包括書名、作者、出版社以及 ISBN 書號。例如：「*Making Things See*, by Greg Borenstein (O'Reilly). Copyright 2012 Greg Borenstein，978-1-449-30707-3.」

如果你覺得你的範例程式碼使用逾越了合理或授權的範圍，歡迎與我們聯繫，來信請寄 *permissions@oreilly.com*。

4

骨架資料

本章中，我們將會學習如何使用 OpenNI 的使用者追蹤資料。到目前為止，我們一直都是直接使用來自 Kinect 的深度資料。為了讓我們的程式具備互動性，我們必須處理此深度資料，以便回應使用者的操作。因為我們無法直接知道使用者的位置，我們不得不由深度點的位置來推斷它。第 2 章中，我們透過追蹤「與 Kinect 的距離最近點」來進行此事。上一章，我們透過在特定的立方體空間中尋找深度點來進行此事。

不過，從現在開始，我們會採取更直接的途徑。OpenNI 有能力為我們處理深度影像，讓我們得以偵測及追蹤人體。因此我們不必透過迭代深度點來檢查它們的位置，我們可以直接透過 OpenNI 追蹤每個使者身體上之每個部分的位置。一旦 OpenNI 偵測到一個使用者，它將會告訴我們使用者之每個可見關節的位置：頭部、頸部、肩部、肘部、手部、軀幹、臀部、膝蓋及腳部。（此清單所列舉的關節不具備解剖學上的意義；OpenNI 使用關節（*joint*）這個術語來指稱程式庫能夠追蹤之使用者身體上的各個點，不論它們是否真的是關節。）

為了建構使用者可以透過身體控制的互動程式，這正是我們需要的資訊。讓我們回頭想想看第 2 章所介紹之「無形的鉛筆」應用程式。如果我們能夠取得使用者之手部的位置，該程式將會變得更易於開發並且更穩固。也就是說，我們不必計算深度影像的臨界值，然後迭代其中的每個點以便找到距離最近點，以及假設那就是使用者伸出的手，我們只需要從 OpenNI 取得使用者之手部的位置，然後使用它來更新線段的位置。此外，使用關節資料，我們將能夠實作出比之前還複雜的使用者介面。我們可以根據手勢及身體的姿勢來進行互動，我們也可以追蹤身體的移動並比較前後的差異，我們還可以量測身體各部分之間的距離。

本章內容

如果真的那麼有效，為什麼我們不一開始就使用關節資料？為什麼我要用兩章的篇幅來探討深度資料？因為關節資料的使用比我們目前為止所看到的深度影像及點雲還要複雜的多。不具備基本的程式設計能力，包括你在前兩章所學到的陣列操作以及 3D 圖形，想要使用來自 Kinect 的關節資料，可說是相當困難。此外，要在應用程式中使用關節資料，我們必須學習一系列新的程式設計技巧。首先，我們必須學習 SimpleOpenNI 所提供之用於存取關節資料的函式。與本書中我們到目前為止所用過的 Processing 程式碼相比，這些函式更為複雜且更難以使用。除了這些基本的知識，要從 SimpleOpenNI 取出關節資料，我們必須掌握 *setter*（設值）方法及 *callback*（回呼函式）之類的新程式設計概念。此外，我們必須學習一些新的數學技法。前一章，我們學到了在三維空間中進行導航及描繪的基礎知識，我們還花了許多功夫在向量的理解及處理上。現在，為了比較三維空間中的各個點，並分析各個點的移動，我們將需要學習如何在三維空間中進行向量的計算。這將會涉及向量減法、點積、叉積之類的數學技法，以及用於計算向量的其他工具，而不單單是把它們當成點來描繪。

本章中，我會一次介紹一個技法。你將會學到許多東西，而且會學得很快，但是我會盡可能讓一切清晰且直觀。與之前各章相比，本章不會對程式碼做完整的描述，而且需要學一點數學知識，但如果你看過了之前各章的內容，那麼你已經做好了準備。然而回報將是巨大的；讀過本章之後，你將有能力創建各種引人注目的互動應用程式。

首先我們將學習用於存取關節資料之 SimpleOpenNI 函式的規則。我們會撰寫一支程式；它會偵測單一使用者，而且會根據使用者手部的位置在螢幕上移動一個圓點。在此過程中，我們將會探討標定使用者所必須進行的校準程序（calibration procedure），以便使用關節資料。我們將會探討何以會存在此程序，以及如何在程式碼中進行此程序。為了存取關節資料，我們將與 OpenNI 的回呼函式做第一次接觸。就像 Processing 本身提供給我們的回呼函式，讓我們得以輕易撰寫出，使用者單擊滑鼠或壓下按鍵時所執行的程式碼，SimpleOpenNI 本身所提供的回呼函式，讓我們得以在使用者追蹤過程中的關鍵時刻（例如，當一個新使用者被發現的時候、當首次開始追蹤的時候、當一個使用者不見的時候…等等）執行執行碼。學習如何使用這些回呼函式是能夠充分利用 OpenNI 之追蹤能力的關鍵所在。

一旦我們學會單個關節的存取之後，我們將把我們的程式擴展成能夠存取使用者骨架中的所有關節。我們將使用此資料繪製出會跟隨使用者姿勢的火柴人（stickman）。SimpleOpenNI 提供了一組輔助函式，以便協助我們繪製「肢體」，也就是將所偵測到的各個關節連接起來的直線。一旦我們讓火柴人出現在螢幕上之後，接著我們會上第一堂的 OpenNI 剖析課。我們將看到我們可以存取的每個關節，以及它們

代表了使用者身體的哪個部分。你將發現 OpenNI 的解剖課與醫師所上的有些不同。

學會如何存取使用者的整個骨架後，我們將能夠著手讓我們的應用程式具互動性。本章接下來的幾節，將側重在可用於分析使用者關節之位置及移動的技術上。我們將從最簡單的技術開始：量測兩個關節之間的距離。為了做到這一點，我們將需要多瞭解一些與向量有關的知識。我們將看到向量如何能夠表示空間中的各個點，以及兩點之間的距離，而且我們將會學習如何透過一些基本的向量減法來量測距離。

接著我們會探討使用者肢體之間的夾角。要描述個別的姿勢，這些夾角是不可或缺的。例如，你如何判斷使用者現在是坐著的？一個重要的線索是，她的小腿與大腿之間的夾角接近 90 度。為了計算兩個向量之間的夾角，我們會使用一個稱為**點積**（*dot product*）數學技法。我將會告訴你，如何計算兩個向量之間的點積及**叉積**（*cross product*）以便找出它們之間的夾角。這些技法還有另一個有趣的用途，它們讓我們得以將一個物體的方向搭配上一個向量，這為我們提供了一個可用於操作 3D 模型的強大新工具。前一章中，我們學到了如何在點雲之內顯示 3D 模型。本章中，我們將會把 3D 模型視為場景的一部分。我們會在我們的座標系統之內設置它們，然後會在它們的周圍移動攝影機。我們將會從不同的角度來觀看它們，但是它們與座標系統本身的相對位置維持不變。有了這個新能力，我們可以使用一個向量來設定另一個向量的方向，這讓我們得以移動這些 3D 模型。我們可以把 3D 模型的方向映射至使用者之一或多個肢體的方向，例如，當使用者移動他的雙臂時，我們可以讓他身後之 3D 模型的翅膀上下舞動。

這些使用來自 Kinect 之骨架資料的技術，形成了用於創建使用者介面的最有力基石。然而，OpenNI 還有一些其他的技巧可以不需要進行完整的校準程序。這些技巧仍舊會使用 OpenNI 的能力，來分析深度影像以及偵測其中的人體。知道如何使用它們可以補充我們已經學到的部分，這讓我們得以有額外的工具來分析場景以及偵測使用者的動作。讀過本章之後，我們將會學到三種技術：場景分析、手部追蹤以及找到使用者的質心。

本章一開始，我有提到 OpenNI 會透過處理 Kinect 所捕捉到的深度影像來產生骨架資料。嗯，除了使用者之關節的位置，OpenNI 還可以從深度影像取出其他資訊。首先，OpenNI 可以判斷深度影像中哪些像素代表人體，以及哪些像素代表場景的背景。只要 OpenNI 一偵測到有人進入場景，此資訊就會變得可供存取。OpenNI 的演算法會偵測是否有人存在，並在她四處移動時不斷追蹤她，所以我們可以區分個別的使用者以及背景。OpenNI 將以「映射」（map）的形式

來提供哪些像素屬於使用者的相關資訊；這是一個與深度影像中每個像素相對應的數字陣列。若像素是背景的一部分，則每個像素的映射值將會是 0。若像素是某個使用者的一部分，則映射值將會是該使用者的識別碼：1、2、3 …等等。本章中，我將會告訴你如何使用此映射值來實現一種稱為**背景去除**（*background removal*）的技術。背景去除是在影像不改變的情況下，從背景中分離出前景元素（例如人體）的過程。它的工作原理就像電影特效中的藍幕去背（green screen process）。我們可以使用背景去除讓你看起來像是站在異國情調的地點，即使當時你是站在房間中你的 Kinect 前面。

除了從場景的背景中分離出使用者，OpenNI 還可以分析背景本身，區分出其中不同的物體，像是傢俱及牆壁。這可能遠不及使用者追蹤可靠（畢竟，Kinect 是被設計來為遊戲進行使用者追蹤的 — 這些其他的場景分析能力只是幸運的副作用）。也就是說，要產生類似建築圖的場景分解圖，它仍然可以派上用場。我將告訴你如何存取這個經展開的場景映射，以及如何將它的資料顯示在標準的深度影像上。

最後，我們將被動學習幾個追蹤使用者的訣竅。即使沒有進行校準程序，OpenNI 仍能偵測你手部的位置及移動。對於因為永久性衰弱、或是因為自己當前的位置或所做的事情，無法移動自己身體的人，這是一個特別有用的介面。我將告訴你如何利用 OpenNI 內建的手勢辨識系統，來偵測場景中是否出現使用者的手，以及如何使用相對應的回呼函式來追蹤它。雖然骨架及手部的追蹤對於在旁邊積極參與應用程式的人來說極為有用，但是在距離較遠處的路人呢，他們甚至可能不知道有這個應用程式的存在？ OpenNI 為我們提供了一個函式，讓我們也能追蹤他們。只要 OpenNI 偵測到有任何人出現在場景中時，它將會開始回報他們的**質心**（*center of mass*）。我們可以使用這些點來追蹤他們，並讓我們的應用程式回應他們，不管他們是否知道它的存在。

為了把這些新的資訊轉化為實際的行動，本章將會介紹兩個完整的專題。相較於前兩章的專題，這兩個專題將會稍微複雜一些。既然你已掌握了這麼多的技能，我相信你已具備這樣的能力。加上關節資料之後相當有用，它讓我們有機會建立一些更實用的應用程式。結果，與之前我們看過的專題相比，本章的專題將會稍微長一些，但是其中也將包括更實用的程式碼，你將能夠把它們重用在本書以外你自己的專題中。

在第一個專題中，我們將會建構一個程式，它能夠對你的日常鍛鍊提供回應。例如，為了協助你鍛鍊身體，體能教練或職業治療師將會親自示範給你看。然後，當你進行鍛鍊時，你會盡力複製教練的動作，

她將會給你回應，告訴你應該伸展地更遠或是彎曲地更深，或是當你做對了也會讓你知道。

此程式將可扮演教練的角色，在你的教練離開時，將她的指示提供給你。我們首先會把特定運動的一系列動作預錄下來。這涉及了將每個關節的位置隨時間的變動，儲存成你所要完成的動作。然後，我們會播放這些動作。我們將會使用預錄下來的關節位置變動來產生火柴人動畫，以便示範最初的正確動作。我將會介紹我自己所創建的一個 Processing 程式庫，透過它我們輕易就能完成關節資料的預錄及播放工作，而且我會對它的工作原理做些說明。

此刻，我們所擁有的是更廣泛有用的程式碼，而不只是這個與運動有關的應用程式。預錄及播放關節資料的這個能力對各式各樣的事物都很有用，像是簡單的行動捕捉（例如，將它與我之前提到的 3D 物體動畫製作軟體結合在一起）或是教學系統。

一旦我們的火柴人能夠演示理想版本的動作時，最後的步驟將會是以這個理想版本的動作，來比較使用者實際的動作。我們將會提供使用者即時的回應，好讓他知道他的動作與理想的版本之間有何差異，以及他要做什麼來修正他的動作。為了做到這一點，我們將會用到稍早我們在比較向量的章節所學到的許多知識，我們將會在每個畫面上量測理想關節位置與使用者實際位置之間的距離及角度。我們將會畫出連接的向量，藉此讓使用者知道如何修正他的動作，以接近理想的版本。

在我們的第二個專題中，我們的重點不再是移動的關節位置，而是靜態的姿勢。我們將建構一支程式，它會偵測著名的舞蹈姿勢，而且會利用它們來觸動適當伴奏音樂。例如，你可以使用 Saturday Night Fever（週末夜狂熱）的招牌姿勢：突出臀部、手指頭指向天空。第一個專題的程式主要是在絕對空間中比較兩個動作是否相符，然而第二個專題的程式主要是在任何特定的時刻比較關節之間的相對位置。我們的程式如何能夠知道你所擺出的是 John Travolta 在 Saturday Night Fever 中的經典姿勢？關鍵在於你的關節之間的相對關係。你的右手是否高於你的頭部？你的左手是否在你的左臀附近？你的左上臂與你的左前臂之間的夾角是多少？如果我們把一個姿勢定義成一組這樣的關係，那麼我們就可以在使用者實現它們時偵測出來，而且我們可以透過觸動相對應的音樂做為獎勵（當然，我們已經從〈透過 Minim 加入聲音〉知道如何播放音訊檔）。此專題的靈感來自 Alex Vessels 與 Kimi Spencer（紐約大學互動電子通訊研究所的學生）的作品以及《Michael Jackson Experience》（這是 Ubisoft 為使用 Kinect 的 Xbox 360 所設計的遊戲，以便測試玩家是否有能力跳出 Michael Jackson 的舞步）。

看過本章之後，你將能夠使用來自 OpenNI 的關節資料，建構出你自己的互動程式，讓使用者能夠透過自己的身體來控制它。你將能夠以火柴人的形式，在螢幕上顯示使用者的身體位置，並且能夠使用他的動作來操縱 3D 物體。你將能夠預錄使用者的動作以便稍後播放。你將能夠量測使用者身體各個部分之間的距離及夾角。你將能夠描述不同的姿勢，並在使用者做到的時候偵測出來。

但是在你能夠做這件酷炫的事情之前，你首先必須能夠存取單一使用者的單一關節位置。所以，讓我們開始吧。

前面幾章的程式碼，一開始總是很簡單，然而在本章中，即使是最簡單的關節追蹤程式也會變得很複雜。關節追蹤資料的使用涉及了一系列完全無法避免的必要步驟。

首先是把用於處理「使用者偵測生命週期」（user-detection life cycle）的所有回呼函式加入我們的程式。隨著使用者出現在攝影機前面展現自己的蹤跡，他將會被偵測並被追蹤。SimpleOpenNI 將會觸動一系列必要的函式，讓我們知道所經過的每個步驟及其結果。在這些函式中，我們必須進行特定的操作，以便讓偵測及追蹤的過程繼續下去。整個偵測過程以非同步方式發生，使得事情變得更複雜。換句話說，主要的 `draw` 函式之執行是並行的。這意味著，除了這些回呼函式的處理，我們還必須考慮到 `draw` 之內的程式碼於執行期間是否會偵測到使用者。

值得慶幸的是，在你瞭解這些步驟之後，你只要透過複製及貼上的方式，就可以把它們加入你未來的程式。舞蹈姿勢的回呼函式，如同每一個會進行使用者追蹤的程式。然而，瞭解每一個舞姿仍然很重要。需要使用者展現自己的蹤跡，並著手為 OpenNI 進行必要的校準程序，以便鎖定他，這是一個複雜的使用者介面設計。根據你的程式的用途，你可能需要提供使用者不同的指示及回應。舉例來說，如果使用者可以看螢幕，你可以使用圖形及顏色來顯示他應該擺出什麼姿勢，並在他被成功追蹤時提供回應。但是，你的程式也可能不會有任何視覺介面。因為使用者可以透過自己的身體提供輸入給 Kinect 應用程式，所以聲音的回應可能會比較適合你的程式。無論採取哪種方式，你都必須瞭解追蹤流程中每個回呼函式意義，以及何時提供何種回應給使用者。

警告過你困難的部分之後，現在讓我們來看看關節追蹤的實際情況。我將會列示程式碼，說明如何使用它，然後我們會對它進行探討。在這個過程中，你將會學到如何使用我介紹給你的這些回呼函式，而且我會指出一些其他的技巧及怪癖。

下面是程式碼：

```
import SimpleOpenNI.*;
SimpleOpenNI  kinect;

void setup() {
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();

  // 開啟使用者追蹤功能
  kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);

  size(640, 480);
  fill(255,0,0);
}

void draw() {
  kinect.update();
  PImage depth = kinect.depthImage();
  image(depth, 0, 0);

  // 建立整數向量以便儲存使用者清單
  IntVector userList = new IntVector();

  // 將使用者清單
  // 寫入我們的向量
  kinect.getUsers(userList);

  // 如果我們找到任何使用者
  if (userList.size() > 0) {
    // 取得第一個使用者
    int userId = userList.get(0);

    // 如果我們校準成功
    if ( kinect.isTrackingSkeleton(userId)) {
      // 建立一個向量來儲存左手的位置
      PVector rightHand = new PVector();
      // 將左手的位置放入該向量
      kinect.getJointPositionSkeleton(userId,
                SimpleOpenNI.SKEL_LEFT_HAND,
                rightHand);

      // 將所偵測到的手部位置
      // 轉換成「投影」座標
      // 以便與深度影像相符
      PVector convertedRightHand = new PVector();
      kinect.convertRealWorldToProjective(rightHand, convertedRightHand);
      // 並顯示它
      ellipse(convertedRightHand.x, convertedRightHand.y, 10, 10);
    }
  }
}

// 使用者追蹤回呼函式！
void onNewUser(int userId) {
  println("start pose detection");
  kinect.startPoseDetection("Psi", userId);
}
```

```
void onEndCalibration(int userId, boolean successful) {
  if (successful) {
    println("  User calibrated !!!");
    kinect.startTrackingSkeleton(userId);
  } else {
    println("  Failed to calibrate user !!!");
    kinect.startPoseDetection("Psi", userId);
  }
}

void onStartPose(String pose, int userId) {
  println("Started pose for user");
  kinect.stopPoseDetection(userId);
  kinect.requestCalibrationSkeleton(userId, true);
}
```

此程式實作了進行使用者追蹤的必要步驟。一旦有一個使用者被偵測到並被校準成功，它會使用關節位置資料讓一個紅色的圓點追蹤使用者的右手。

第一個步驟是告訴 OpenNI，我們想要開啟使用者追蹤功能。此事會在 `setup` 之內的這一列發生：`kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL)`。正如之前，當我們想要存取來自 Kinect 的深度及彩色影像時，我們會叫用 `kinect.enableDepth` 及 `kinect.enableRGB`，現在我們會告訴 OpenNI，我們還想要存取骨架資料。不同於其他的啟用函式，這個函式需要一個引數：`SimpleOpenNI.SKEL_PROFILE_ALL`。`SimpleOpenNI.SKEL_PROFILE_ALL` 是一個常數，用於告訴 SimpleOpenNI 我們想要追蹤使用者之骨架中的所有關節。還有一些其他選項可用，但在幾乎所有情況下，你將會想要追蹤完整的骨架，所以當你叫用 `enableUser` 時，你總會傳入此引數。

當我們要來瞭解如何處理使用者追蹤資料時，除了 `setup` 中的這一列程式碼，我們還會看到許多新的程式碼。繼續閱讀下去，我們將會把此程式中的程式碼整個看過，但首先你應該實際執行看看以及學習如何讓 OpenNI 追蹤你。當你執行此程式時，你所看到的畫面有如一般的深度影像。一旦此程式偵測到了一個使用者，它將只會在深度影像上進行描繪的工作。不過，在任何令人興奮的事情發生之前，你必須進行校準的工作。

注意校準的問題

為了讓 OpenNI 的演算法開始追蹤一個人的關節，這個人需要擺出已知的姿勢。具體來說，你必須雙腳站在一起，而且你的雙臂高舉超過你頭兩側的肩膀。這個姿勢有很多名稱，在技術文獻及 PrimeSense 自己的文件中，這稱為“Psi”姿勢。有的設計者稱此為“submissive”（順從）姿勢，因為這非常像有人拿槍指著你時所擺的姿勢。

不管名稱是什麼，校準姿勢（calibration pose）總是會引發使用 Kinect 之設計者的熱烈討論。有時這會很不方便，如果你想在人們進行正常活動的時候追蹤他們，要求他們停下來進行明確的校準操作，是一個難以越過的障礙，但是，有些人則擔心自己可能會在不知情的狀況下遭到追蹤。因為人體追蹤攝影機可能會在你不知情的狀況下觀察到你的每個動作，所以當你在公共空間的時候，你就會有這樣的疑慮。

無論你的應用程式或是你如何理解這些問題，它們都應該得到注意。你可能會驚訝使用者對校準程序有如此強列的反應。

然而，這個問題有一個可能的解決方案。在某些情況下，OpenNI 不必進行明確的校準程序，就能夠追蹤使用者。OpenNI 可以記錄來自單一使用者的校準資料，然後使用類似的身體姿態來校準其他使用者。此技術有些複雜，而且並非任何情況都適用 — 舉例來說，如果你預期你的應用程式的使用者包括小孩及成人、男人及女人…等等。由於太過複雜，本章將不會探討如何使用被儲存下來的校準資料。然而，我在附錄中有提供兩個範例程式，用於示範此過程。第一支程式（範例 A-3）將會記錄一個 *.skel* 校準檔，第二支程式（範例 A-4）將會載入該檔案以取代平常的校準程序。

因此，校準程序實際上要如何進行？身為使用者，我們該怎麼做才能讓 OpenNI 來追蹤我們？我們必須撰寫什麼樣的程式碼才能讓此過程發生？讓我們來逐步檢視校準的過程，看看使用者所採取的動作以及所執行的是程式中的哪些部分。

校準程序的各個階段

在我們探討程式碼的具體細節之前，我想讓你們先看一下校準程序的概圖（圖 4-1）。此圖可以看到 OpenNI 觸動回呼函式的流程，以及為了讓校準程序繼續下去，我們必須採取的動作。我會簡單說明此概圖，然後我們會深入探討實際的程式碼，以便進一步瞭解整個系統的實際工作方式。

校準程序是我們的程式與 OpenNI 之間來回進行的過程。每個步驟中，OpenNI 將會觸動一個回呼函式。然後，在該函式中，我們需要採取一個行動，以便繼續此過程。此動作將會依序觸動下一個回呼函式…等等。

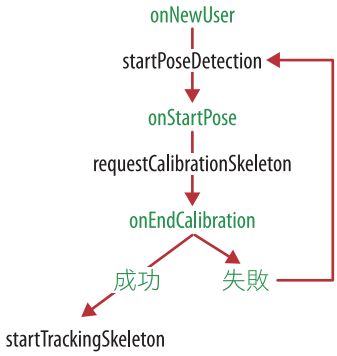

圖 4-1：為了初始化關節追蹤，所需進行的 OpenNI 行動及回呼函式。

追蹤的三個階段分別是偵測到使用者、進行校準以及追蹤到使用者。當此程式開始執行時，OpenNI 將不會追蹤任何使用者。它所提供用於存取使用者的函式，將會傳回一個空清單。當有一個使用者進入場景時，就會開始這個過程。於是 OpenNI 會偵測到使用者的存在，並以該使用者為追蹤候選人。它會叫用我們的程式所提供的 `onNewUser` 函式。在該函式裡，我們會透過 `startPoseDetection` 的叫用來初始化追蹤過程。此函式會要求 OpenNI 去檢視使用者，看看她是否擺出了校準姿勢。一旦我們偵測到一個使用者，`draw` 函式中的使用者清單將會被填值。然而，在校準程序完成之前，這些使用者的關節資料尚不存在。一段時間後，只要使用者擺出校準姿勢，OpenNI 將會叫用我們的程式所提供的 `onStartPose` 函式。接著，為了繼續校準程序，我們需要叫用 `requestCalibrationSkeleton`，此函式會觸動 OpenNI 的實際骨架偵測過程。當此過程完成時，OpenNI 將會透過叫用另一個回呼函式（我們的程式所提供的 `onEndCalibration` 函式）來回報此情況。校準程序不一定會成功，在此回呼函式裡，OpenNI 將會回報校準的狀態。如果校準成功，我們會叫用 `startTrackingSkeleton` 以便著手存取來自使用者的關節資料。如果校準失敗，我們會透過再次叫用 `startPoseDetection` 來重新開始此過程。在我們這麼做以提高校準成功的機會之前，我們可能會想要在螢幕上提供額外的回應給使用者。

所以，完整的步驟就是：偵測到使用者之後予以校準，最後進行關節追蹤的初始化。現在我們將更深入探究每個步驟，以便檢視實作它的程式碼，包括這些回呼函式中的程式碼，以及我們在 `draw` 中使用骨架資料之處。

使用者偵測

正如我提到的，當使用者進入深度攝影機的視野才會開始此一過程。當使用者出現時，OpenNI 會意識到使用者的存在。此時，使用者尚未被校準，他的關節資料尚不存在。OpenNI 基本上只是意識到有人

在深度攝影機的視野中四處走動，OpenNI 會叫用我們的程式所提供的 onNewUser 函式。

根據實際情況的不同，這可能發生地比你預期的還快。就算使用者的身體進入畫面的部分並不多，OpenNI 也能夠偵測到它的存在。例如，圖 4-2 所示為此程式的螢幕截圖，圖中可以看到 OpenNI 首次偵測到我的實況。你可以看到我剛剛進入畫面左側的情形，圖中只看到了我的手臂以及我的胸口的一部分。

讓我們來看看此回呼函式的程式碼。當 OpenNI 偵測到一個新的使用者時，我們的程式會做哪些事？

```
void onNewUser(int userId) {
  println(“start pose detection”); ❶
  kinect.startPoseDetection(“Psi”, userId); ❷
}
```

圖 4-2：OpenNI 可以在非常早期的時候確認場景中的使用者。此螢幕截圖取自 onNewUser 回呼函式執行的時候，你可以看到我剛剛進入畫面左側的情況。

首先要注意的是 onNewUser 的引數；這是一個整數，代表被偵測到之使用者的識別碼。OpenNI 可以同時追蹤多個使用者，所有使用者都會被賦予獨一無二的識別碼，讓你得以區分所要追蹤的每一個使用者。就我的經驗來說，會同時追蹤多個使用者的應用程式相當少見。然而，如果你希望你的程式追蹤兩個人，並讓他們扮演不同的角色，你可能需要使用用「使用者識別碼」（user ID）。例如，可能你會想讓第一個出現的使用者控制螢幕上代表她的一個特定圖形，並讓第二個使用者控制另一個特定圖形。要做到這一點，你只需要使用每個被追蹤使用者的識別碼，就可以在你的 draw 函式裡將圖形擺在適當的位置。