僑光學報　第 43 期　頁 1-15　中華民國一〇九年十月

# 以補間動畫實作碰撞偵測

胡興民

## 摘　　要

判別兩物件的碰撞測試不但應用於電腦動畫、遊戲場域，亦已應用於虛擬實境(VR)或擴增實境(AR)的使用者(user)或滑鼠與物件是否發生接觸的依據；更是網路3D瀏覽器判別接觸的利器，以禁制瀏覽人穿透(walking through)物件的不真行為。一般採行矩形拘束框(bounding shape)或圓形範圍框作為碰撞的測試依據。本文先行探索球面體(sphere)或匣面體(box)與無限平面(infinite plane)之間的碰撞問題，接著探討更實際的情形--即球面與任意環境的碰撞偵測；藉由導入齊次座標與四維向量，再借助射影變換以解得碰撞發生時以及碰撞點的向量通解。最後以MATLAB程式語言實作移動剛體(rigid)的碰撞測試，並以給定的直線及曲線模擬移動補間動畫；同時也把重力影響速度的快慢設計在程式內。實作困難處在於碰撞角度的計算與碰撞後的移動方向，甚至碰撞後的移動速度或elastic object的瞬間變形樣貌等。

關鍵詞：碰撞偵測、射影變換、補間動畫

ottohu@mail.tf.edu.tw
東方設計大學 遊戲與動畫設計系 講師

# 壹、簡 介

### 1.1 碰撞偵測與補間動畫(Tweened Animation)

物件的碰撞偵測是電腦遊戲的核心功能之一，其可增進電腦遊戲的互動性與衍生多面向的其他功能，例如發生碰撞而衍生運動方式的改變(彈跳或轉彎方位以及大小、變形否等等)或子彈發射後發生碰撞而衍生爆炸、衍生防護罩的出現或衍生的回擊等反應。[1]可瞭解碰撞的變形期與恢復期以及相關概念。當電腦遊戲中的移動物件彼此之間或與其周遭環境之間互動時，在相同的時間點、它們常會企圖佔有相同的空間場域，此時即發生所謂的碰撞(collision)。由於物件可能擁有複雜的幾何形狀，一般為減輕碰撞偵測的計算負荷，大一些的物件多以「有界體」(bounding volume、也稱範圍框，如圖一)近似物件以利碰撞計算；非常小的移動物件常以「點」代表物件，因而碰撞偵測變成射影幾何(ray intersection)計算的相交問題。物件的範圍框在一般的遊戲場域中是看不到的(除非刻意另行操作)，偵測碰撞與否即為檢測不同物件的範圍框是否相交[10]，圖一列舉物件的不同範圍框。

另外，由於畫面(frame)之間的間隔時間非常短，一般假設物件在畫面的間隔期間是直線移動的，即使已知該物件是沿著曲線行進時亦然；因此偵測物件是否發生碰撞又變成判別物件表面沿著某一線段的推出點(extrusion)，是否相交到其它物件或周遭環境的某個部件(即某個部份)。

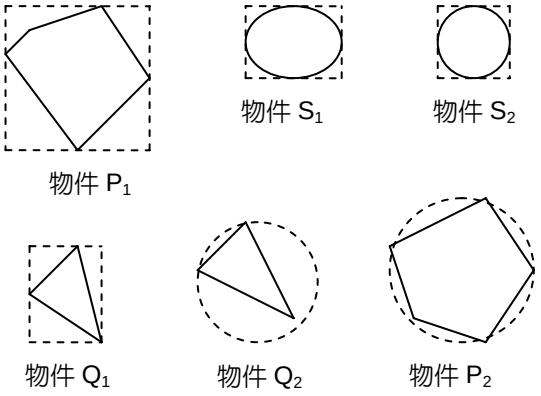
本文先行探索球面體(sphere)或匣面體(box)與無限平面(infinite plane)之間的碰撞問題，接著探討更實際的情形--即球面與任意環境的碰撞偵測；最後以MATLAB程式語言實作移動剛體(rigid)的碰撞測試，並以給定的直線及曲線模擬移動補間動畫；同時也把重力影響速度的快慢設計在程式內。實作困難處在於碰撞角度的計算與碰撞後的移動方向，甚至碰撞後的移動速度或elastic object的瞬間變形樣貌等。

自Flash5.5起的腳本程式語言ActionScript(AS3.0)已把AS2.0的碰撞偵測由原本的hitTest函式，進階並細分成更精準的hitTestPoint函式與hitTestObject函式[3]；hitTestObject函式執行剛體(rigid)物件範圍框對另一物件範圍框的碰撞測試，而hitTestPoint函式執行物件範圍框對另一物件內所含之物件的「點」碰撞偵測。另述在「貳、四、實作(1)」。

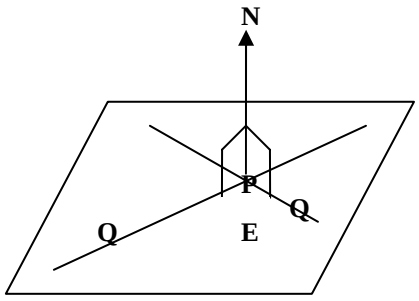
補間(tween)源自於傳統動畫製作領域，資深動畫設計師負責把人物或角色的最初與最後動作、姿態等先繪製出來，資淺動畫設計師再完成"填補中間過程"(in-between)或是進行期間的過程處理作業。移動補間動畫是針對不同畫面中的某個物件屬性(例如位置、顏色等)指定不同值所建立的動畫，是建立運動的有效方法，且此屬性可隨著時間變化；以 Flash 而言、指定不同值後，Flash 會自動接手計算該屬性在兩個畫面之間的值。

一般動畫軟體多已內建貝茲曲線(Bezier)、較少內建雲形線(B-spline、或稱條形曲線)作移動補間，只須更動其調整點或稱節點即可完成移動補間動畫。本文著重學理闡釋，故自行設計移動補間的直線與曲線。

### 1.2 球面與平面之間的碰撞




圖一 例示矩形範圍框(虛線)與圓形範圍框




圖二 給定法向量 N 與點 P，則一平面 E 由滿足(Q – P) ⊥N 的所有 Q 點形成。

給定法向量$\mathbf{N}^1(A,B,C)$與空間中的一點$\mathbf{P}(s,k,u)$，則滿足 $(\mathbf{Q} - \mathbf{P}) \perp \mathbf{N}$ 的所有$\mathbf{Q}(x,y,z)$ 點形成一平面 **E**（圖二），即 $\mathbf{N} \cdot (\mathbf{Q} - \mathbf{P}) = 0$ 所以 $A \cdot (x - s) + B \cdot (y - k) + C \cdot$

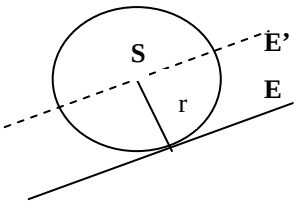
$(z - u) = 0$ ，整理後得Ax + By + Cz – (As + Bk + Cu) = 0；平面方程式一般表示成Ax + By + Cz + D= 0 其中 $D = -\mathbf{N} \cdot \mathbf{P}$，而 $|D|/\|\mathbf{N}\|$ 為過原點**O**之另一平行平面相對於此平面的位移距離。圖三例示一球面(其球心**S**)與一平面**E**接觸時，球心**S**到平面的距離為 $r = \mathbf{E} \cdot \mathbf{S}$；若將平面**E**寫成四維向量、即 $\mathbf{E} = [\mathbf{N}, D]$，則 $r = \mathbf{E} \cdot \mathbf{S} = \mathbf{N} \cdot \mathbf{S} + D$ 或 $\mathbf{N} \cdot \mathbf{S} + D - r = 0$；後者表示點**S**位於平面$\mathbf{E'} = [\mathbf{N}, D - r]$，此平面**E'**平行於平面**E**且在平面**E**的法向量**N**的方向位移了距離 r [9]。

假設球心從時間點 t = 0 的位置$\mathbf{S}_0$移動到時間點 t = 1 的位置$\mathbf{S}_1$，並設$\mathbf{S}_0$位於平面**E**之法向量**N**的那一側且與**E**不相交，也就是說 $\mathbf{E} \cdot \mathbf{S}_0 > r$；若 $\mathbf{E} \cdot \mathbf{S}_1 > r$，即可知移動後的點$\mathbf{S}_1$與平面**E**仍不相交。在時間點 t 的球心位置函數**S**(t)可表示成 $\mathbf{S}(t) = \mathbf{S}_0 + t\mathbf{V}$ ，其中**V**為球心的移動速度、即 $\mathbf{V} = \mathbf{S}_1 - \mathbf{S}_0$。若 $\mathbf{E'} \cdot \mathbf{S}(t) = 0$ 則發生碰撞，此時若將位置函數**S**(t)以$\mathbf{S}_0 + t\mathbf{V}$代入$\mathbf{E'} \cdot \mathbf{S}(t) = 0$，可得$\mathbf{E'} \cdot \mathbf{S}_0 + t(\mathbf{E'} \cdot \mathbf{V}) = 0$，解出 $t = -(\mathbf{E'} \cdot \mathbf{S}_0)/(\mathbf{E'} \cdot \mathbf{V})$；因為**V**代表方向且其齊次座標的w座標為0，所以分母又等於 $\mathbf{N} \cdot \mathbf{V}$。若 $\mathbf{N} \cdot \mathbf{V} = 0$則表示球面移動方向平行於平面，故不會發生碰撞；否則就會發生碰撞，其碰撞點**C**(t) = **S**(t) – r**N** (利用位置函數$\mathbf{S}(t) = \mathbf{S}_0 + t\mathbf{V}$可推得，但此碰撞點**C**位於法向量**N**的相反方向之距離 r 處)。

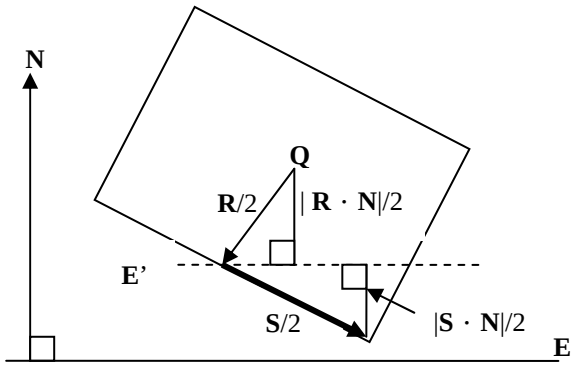
### 1.3 匣面與平面之間的碰撞

匣面(box)與平面之間的碰撞問題除了可能也只接觸於一點(前述球面與平面之間的碰撞就只接觸一點)，還可能碰撞於多點(即不止一點)、例如匣面的某個面恰緊貼於平面。假設一匣面物件的主軸分別為R、S與T，且假設其大小 ‖ R ‖ 、 ‖ S ‖ 與 ‖ T ‖ 恰等於此匣面物件的長、寬、厚之大小；首先決定匣面的等效半徑($r_{eff}$)，以便測試匣面

註1：粗體英文字母代表向量或矩陣，而且P(t)、Q(t)與S(t)等也代表平面或空間中的點。




圖三 半徑為r的球面與一平面E接觸，且球心位於E平面位移r距離後的另一平面E'上。




圖四 平面法向量為N且匣面物件的長、寬、厚分別為R、S、與 T時的等效半徑reff之計算

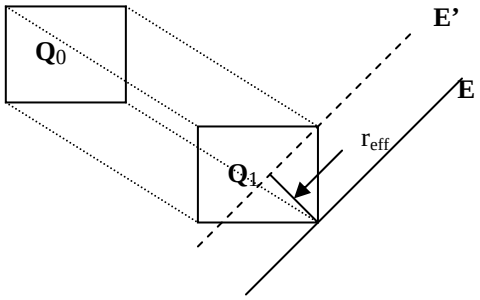
[中心是否位於擴充型視野截錐體[2]之各面裏側。假設平面法向量為N，如圖4所示、等效半徑$r_{eff} = (1/2)(|R \cdot N| + |S \cdot N| + |T \cdot N|)$；若截錐體之各面E表示成四維向量、即E=N,D]，且匣面有界體(匣面範圍框)中心為Q，計算點積、若$E \cdot Q < -r_{eff}$ 則該匣面為此視野截錐體的不可視(not visible)物件(參照圖三及前一節最後一段的說明)[7]。

假設時間點 t = 0 的匣面中心位置Q0移動到時間點 t = 1的位置$Q_1$、如圖五所示，則此匣面中心位置函數Q(t)可表示成$Q(t) = Q_0 + tV$，其中V為匣面的移動速度、即$V = Q_1 - Q_0$。計算$t = -(E' \cdot Q0)/(E' \cdot V)$以求得交點、此時$E' = [N, D - r_{eff}]$；與球面的假設相同、即Q0位於平面E之法向量N的那一側且與E不相交，也就是說$E' \cdot Q_0 > 0$；若$E' \cdot Q_1 > 0$，即可知移動後的點Q1與平面E仍不相交，故沒有發生碰撞。若發生碰撞，就需找出碰撞點；此時如果|R · N、|S · N|與 |T · N| 皆不為零值，則碰撞點即為匣面的某頂點(vertice)。藉由檢視匣面八個頂點的表示式，可以得出所碰撞頂點的位置函數的通式：匣面頂點的位置函數$Z = Q(t) \pm (1/2)(R + S + T)$，點積$E \cdot Z$最小時的頂點就是最接近平面的頂點，亦即$\pm R \cdot N$、$\pm S \cdot N$與$\pm T \cdot N$須全為負值，所以接觸點$C = Q(t) - (1/2)[\mathrm{sgn}(R \cdot N)R + \mathrm{sgn}(S \cdot N)S + \mathrm{sgn}(T \cdot N)T]$。

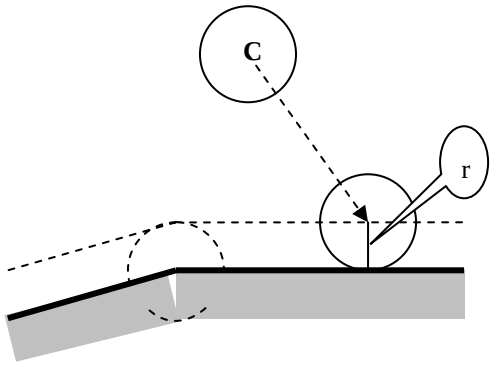
如果這三個值|R · N|、|S · N| 與 |T · N|恰有一為零(值)，則匣面相對應的某一主軸平行於平面，碰撞會發生在匣面的某一邊(或稱某一稜)；例如$|T \cdot N| = 0$，發生碰撞的邊(或稜)上兩端點$C_1$與$C_2$即為取值$\pm T$ 時的 $Ci = Q(t) - (1/2)[\mathrm{sgn}(R \cdot N)R + \mathrm{sgn}(S \cdot N)S \pm T]$、$i = 1, 2$。若|R · N| 、 |S · N| 與 |T · N|這三個值中恰有兩個為零(值)，則碰撞會發生在匣面的某一面(face)；例如 |T · N| =

0 = |S・N|，此時發生碰撞的面上之四個頂點$C_1$、$C_2$、$C_3$與$C_4$即為取值 ±S 及 ±T 時的 $C_i$ = Q(t) – (1/2) [sgn(R・N ) R ± S ± T ]、*i* = 1, 2, 3, 4。

註 2：擴充後的視野截錐體，請參考圖六及其說明；或類似於圖三、圖四中的虛線 E'。也就是將原來的截錐體向外擴充 $r_{eff}$ (等效半徑)後的截錐體；$r_{eff}$ 參考圖三、圖四所述概念。




圖五　將平面位移 $r_{eff}$ 以便反向決定匣面是否碰撞平面




圖六　發生碰撞時，半徑為 r 的球心 C 與擴充 r 單位後的多邊形平面(虛線)相交。

### 1.4 分割的空間之碰撞測試

3D遊戲中常使用八元樹(octree)與BSP tree (Binary Space Partitioning tree)作場景管理或用於建構場景[2][3]。八元樹的定義是：若不為空樹的話，樹中任一節點的子節點恰好只會有八個或零個(也就是子節點不會有0與8以外的數目)；想像一個立方體，最少可以切成多少個相同等分的小立方體？答案是8個。再例如一個房間裡某個角落藏著一枚金幣，如果想很快的把金幣找出來，則可以把房間當成一個立方體，先切成八個小立方體，然後排除掉沒有擺放任何東西的小立方體，再把有可能藏金幣的小立方體繼續切八等份…；如此切割(partitioning)下去，平均在$Log_8k$的時間內就可找到金幣、k為房間內的所有物品數。因此，利用八元樹作3D空間中的場景管理，可以很快地知道物件在3D場景中的位置，或偵測是否與其他物件有碰撞以及是否在可視(visible)範圍內；但，八元樹僅適用在密閉或有限的空間，對於開放式的空間、例如戶外場景，就不太適用。

開放式的空間通常採行BSP tree，它是以平面來分割場景的；場景中每個物件的每個多邊形(polygon)當成一個平面，而每個平面會有正反兩個面，如此就可把場景分成兩部分。先從第一個多邊形平面開始劃分場景，再從這分出的兩部分、個別再以其他多邊形平面繼續細分場景...如此進行，就可把場景建構成一棵二元樹；這樣，很快的就可找出物件的位置，或是否發生碰撞了。

既然八元樹(octree)或BSP tree是以平面將空間分割成不同區域，那麼就可援用前兩節所述概念、取代繁瑣複雜的幾何測試以判別移動物件在同一影格內(frame)是否發生碰撞，也就是$E・Q \geqq r_{eff}$表示物件(即Q點)位於平面E的正向側、$E・Q \leqq -r_{eff}$代表物件位於平面E的另一側。使用此二式可以簡單的方式辨明：若在同一影格內的$Q_1$、$Q_2$同時位於平面的相同側，即可知無碰撞發生。

## II 方法與實作

### 2.1 通用的球面碰撞

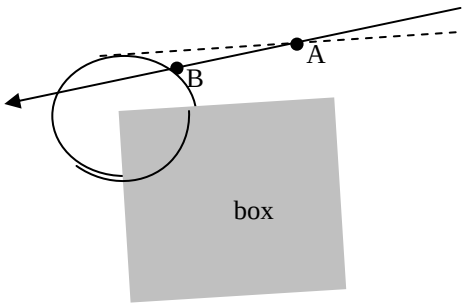
基本的球面碰撞準則：兩個球面體物件發生碰撞時，半徑為r的球面體之球心C位於另一球面體物件擴充r單位後的表面上，此表面即代表會發生碰撞時的球心所在之表面，參照圖6。如果是一球面體物件與多面體物件發生碰撞，則所有球心形成一表面(圖六)，此表面可能有下列三種情形[8][9]：

(1)與多面體物件的某一面發生碰撞時，所有球心形成的表面就是多邊形表面外推(move outward) r個單位後的擴充型多邊形表面；亦即，球心在此擴充型多邊形表面上時，會與多面體的某一面發生碰撞。
(2)與多面體物件的某一稜發生碰撞時，球面體物件之球心位於外推r個單位後的擴充型多面體的稜上；以此外推後的稜為中心線、該球面體之球心並沿著此中心線移動，就形成半徑為r的圓柱體。此圓柱體表面即為會與多面體的某稜發生碰撞時的球心所在之表面。
(3)與多面體物件的某一頂點發生碰撞時，球面體的球心

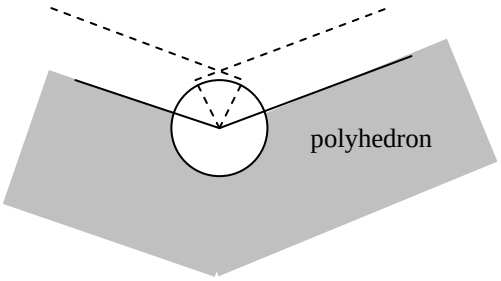
會位於以擴充型多面體之外推頂點當球心、半徑為r的球體表面上；此球體表面即為會與多面體的某頂點發生碰撞時的球心所在之表面。

不論是上述三種情形中的哪一種，皆可用代表球心移動路線的射影光線是否相交到擴充型多面體，作為判別是否發生碰撞的依據。其演算法如下：

(1)先判別球面體物件之球心是否位於擴充型多面體的任一面上。是的話，就略過接下來的(2)與(3)。
(2)再判別球面體之球心是否位於圓柱體的表面上。是的話，就略過檢測(3)。
(3)最後才判別球面體球心是否位於原來的多面體之任一頂點外推後所形成的球體表面上。




圖七　例示一射影線與擴充型圓柱體交於內部 B 前，已先接觸到擴充型多面體的表面 A；後者會先被檢測。




圖八　碰撞偵測時，並非所有的頂點與稜都須被檢測；球體所在凹陷的稜就不須檢測。

如圖七所示，執行上述演算法的(2)時，因為已先行檢測球面體物件是否相交到擴充型多面體的表面內、即步驟(1)，故不須擔心是否會與多面體物件的任一面發生碰撞；同樣地，執行步驟(3)時，也無須再測試球面體物件是否會與多面體物件的任一面或任一稜接觸 (因為已在(1)與(2)檢測過)。

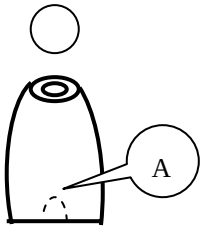
碰撞偵測時，並非所有的頂點與稜都須被檢測；例如圖八中「與某一稜發生碰撞」所建構出的圓柱體之任何部份，都不會對碰撞發生時、球心所在表面有任何影響 (因為此圓柱體完全座落於擴充型多面體內)，故圖八中(球體所在)凹陷的稜就不須檢測會否與該球面體發生碰撞。稜上非端點的頂點，例如圖九瓶罐或類似形體的場景底層內面隆起處的點A，因為已完全在擴充型多面體內，所以不須檢測會否與球面體物件發生碰撞。

**2.2 碰撞後的滑行距離**

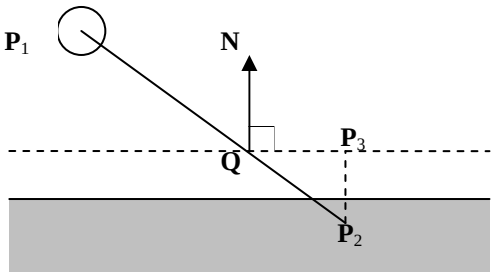
移動物件與靜止的環境場域發生碰撞時，能夠沿著環境幾何的表面滑行(slide)而不卡住，會讓遊戲、動畫更加生動逼真，例如人物、角色(character)的移動。在單一影格內發生碰撞後，物件的新位置由物件與碰撞表面之間的角度決定；假設表面的法向量為N且物件由$P_1$欲行進至$P_2$ (圖十)，因為在Q會發生碰撞，所以未完成路段$\overline{QP_2}$投影到該表面的$P_3$即為碰撞後物件的新位置；碰撞後物件的滑行距離為$\overline{QP_3}$，其算式為$P_3 = P_2 - [(P_2 - Q) \cdot N] N$。[5]內含廣泛的C++演算法與實作。

**2.3 兩個移動球面的碰撞**

假設第一個球面物件在時間點t = 0時的起始位置為$P_1$、在時間點t = 1時的最後位置為$P_2$，另一個球面物件起始位置為$Q_1$、最後位置為$Q_2$ (如圖十一)，並設兩物件皆為定速直線行進且其速度分別為$V_P$與$V_Q$，即$V_P = P_2 - P_1$、$V_Q = Q_2 - Q_1$，兩球心位置函數




圖九　瓶罐或類似形體的場景底層內面隆起處的點A，不須檢測會否與球面體物件發生碰撞。

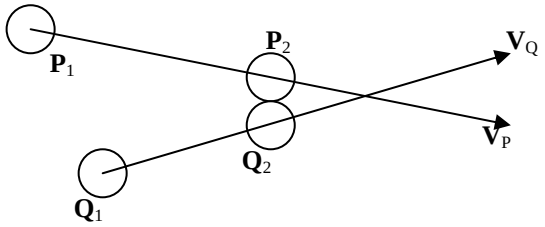



圖十　由$P_1$欲行進至$P_2$，在Q發生碰撞後，物件的新位置在$P_3$。

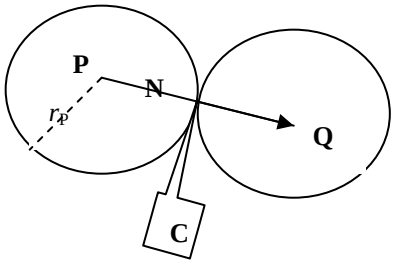
分別為$P(t) = P_1 + tV_P$與$Q(t) = Q_1 + tV_Q$；若$r_P$、$r_Q$分別為兩球面的半徑，碰撞發生時P(t)與Q(t)之間的距離應為$r_P + r_Q = d$。利用距離的平方、即$d^2 = \| P(t) - Q(t) \|^2 = \| P_1 + tV_P - Q_1 - tV_Q \|^2$，並使用符號的代換以利計算、即假設$w = P_1 - Q_1$ 且 $u = V_P - V_Q$，則 $d^2 = \| w + t\,u \|^2 = w^2 + 2\,t\,(w \cdot u) + t^2\,u^2$；利用公式解、得 $t_{1,2} = \dfrac{-\mathbf{w}\cdot\mathbf{u} \pm \sqrt{(\mathbf{w}\cdot\mathbf{u})^2 - \mathbf{u}^2(\mathbf{w}^2 - d^2)}}{\mathbf{u}^2}$。[4]以Radial fields (徑向場)處理複雜物件的變形，可減少pre-computation與記憶體需求，但易流於application-specific (特定應用導向)。

注意$u^2 = 0$ 可能兩物件為靜止狀態或以相同方向相同速度直線行進，因此不會碰撞。根號內若為負值，代表沒有時間點發生碰撞 (沒有碰撞)；兩個t值中的較小t值表示碰撞開始、兩物件愈靠愈近，較大的t值表示碰撞後的分離開始、兩物件會愈離愈遠。碰撞開始的t值如果沒有在0~1個單位時間內，就表示在該時間區間內沒有碰撞；所以類似的計算可以得知在其他的時間區間內有沒有發生碰撞，例如較小t值(碰撞開始的時間點)有沒有落在第二個單位時間到第三個單位時間內(2~3個單位時間內)，或是較小的t值是否介於$t_m$~$t_n$個單位時間內--意即任意兩個不同的時間點內有否發生碰撞。

一旦偵測到碰撞即可計算、找出兩球心位置，如圖十二所示：碰撞點C位於連心線段 $\overline{PQ}$ 上且與球心P(t)距離$r_P$個單位處、即$C = P(t) + {}_{r_P}N$；故C為法向量N的函數，此時的法向量 $\mathbf{N} = \dfrac{\mathbf{Q}(t) - \mathbf{P}(t)}{\|\mathbf{Q}(t) - \mathbf{P}(t)\|}$。




圖十一　兩個移動的球體物件、定速直線行進時的碰撞偵測




圖十二　碰撞點 C 位於連心線上且為法向量 N 的函數

**2.4 實作**

2.4.1 AS2.0的碰撞偵測由原本的hitTest函式，進階並細分成更精準的AS3.0之hitTestPoint函式與hitTestObject函式 [6]，但程式細節屬商業利基，故只能根據其準則(如下)與文中論述的原理另行設計。

(i)HitTestPoint語法通常會根據「判定領域」參數的設定，將X和Y軸座標的物件指定的形狀或者範圍框來加以比較；當兩者實際佔用的區域讓X或Y軸相交於一點，便會判定為true，所以碰撞偵測是在「點」。

(ii)HitTestObject語法則是判斷影片片段和指定物件的範圍框是否有重疊或相交就會傳回true，所以碰撞偵測是在範圍框內的相交區塊上。

2.4.2 演算法：

(i)以MATLAB程式語言配合其"patch"(補丁或貼皮)函式製作剛體並著色，再以射影變換實作剛體移動與碰

撞，程式中也設計有控制移動速度的快慢；MATLAB需設定適當的"Renderer"以及"EraseMode"以得到平順的動畫模擬。

(ii)先設計程式給定直線及曲線，以備模擬電腦動畫中的移動補間動畫 (程式1及圖十三) >再測試行進路線以及碰撞後的移動路徑是否如程式1所設計、規劃的；是的話、即完成移動補間動畫的測試 (程式2及圖十四) >最後實作下降速度較快，碰撞地面反彈後速度變慢的動畫模擬 (程式3及圖十五)。

(iii)程式4假設兩物件大小相同且以質量比、質量差、速度差、位置比與位置差等方式插分計算：不同出發位置、不同質量大小，以及移動方向不同、速度大小也不同的碰撞後之行進角度。可自行輸入不同質量、位置與不同速度以驗證。

2.4.3 操作(optional)：

(i)執行程式1時、出現圖形後，請移動滑鼠到圖中、出現十字型提示標誌後，將十字型標誌的交叉處移近曲線、點按滑鼠右鍵完成圖中文字說明(即軌跡)的設定。

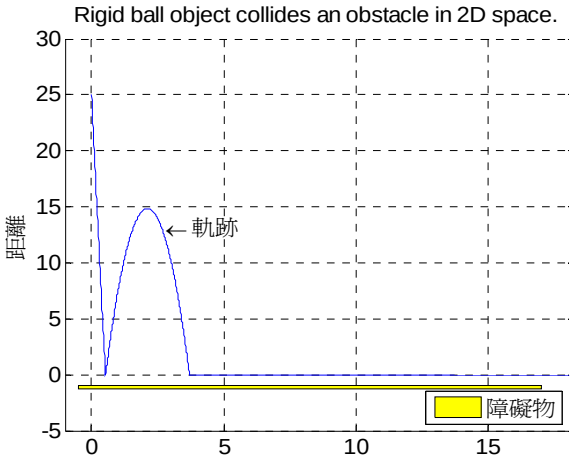
(ii)測試、執行程式2時，畫面會出現“第1次碰撞地面；第3次碰撞地面後，就只在地面移動了”等文字說明。另可同時按下「Ctrl + C」得到動畫的瞬間圖形。

(iii)執行程式3時，可以同時按下「Ctrl + C」得到動畫的瞬間(靜止)圖形。

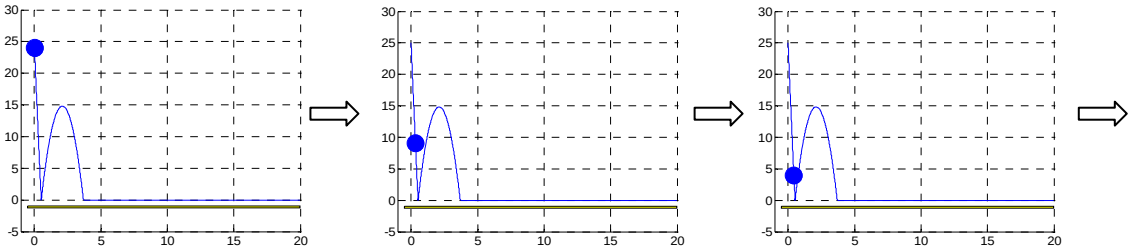
## 參、結果與討論

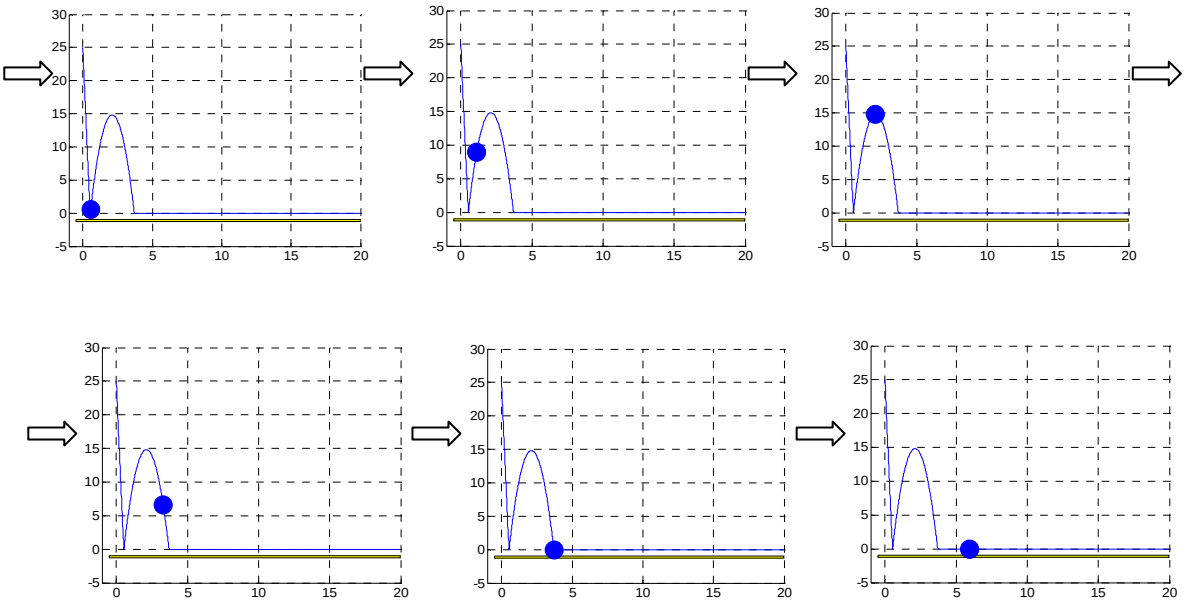
### 3.1 結果

3.1.1 圖十三~十五為程式1~3的輸出情形

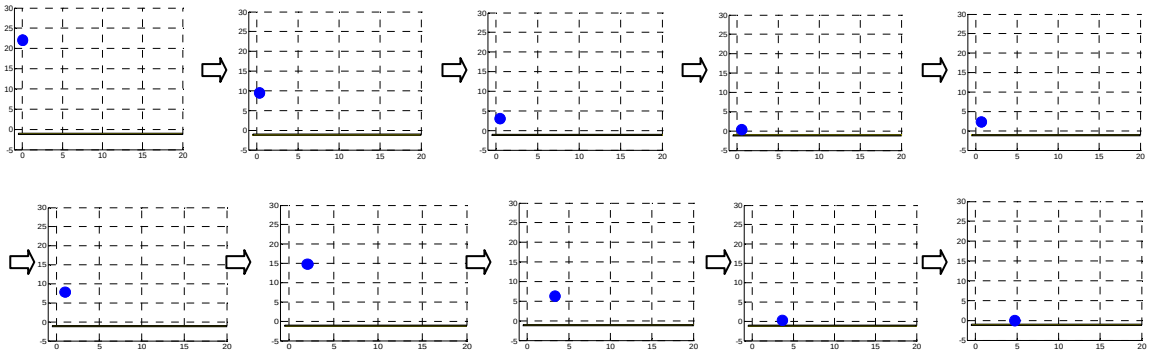



圖十三　規劃(產生)移動補間動畫用的直線與曲線





圖十四　測試碰撞前後的移動路徑是否如移動補間規劃的



圖十五　下降速度較快，碰撞地面反彈後速度變慢的動畫模擬。

3.1.2 程式4驗證碰撞後的行進角度

**r = 1**
**MassCouple1 = 5 1**
**P1 = 0 0**
**P2 = 1 0**

**======= 以下兩物件的出發位置分別固定且半徑相同 =======**
**V1 = 1 1**
**V2 = -1 1**
**狀況1: 質量5:1，兩物件分別以速度向量(1 1)與(-1 1)各自朝右上45度/左上45度移動**
**------- 發生碰撞 -------**
**物件1及2的行進角度分別為7.156505e+001度與2.319859e+001度(後者為23.1986度)**

**MassCouple2 =　1　　1**
**狀況2: 質量比已改變成1:1，速度向量與狀況1相同**
**-------　發生碰撞　-------**
**物件3及4的行進角度分別為135度與45度**

**V3 =　0　　1**
**V4 =　0　　1**
**狀況3: 質量比與狀況2相同(即1:1)，但兩物件分別以速度向量(0 1)與(0 1)(皆)向上移動**
**-------　沒有發生碰撞　-------**
**物件5及6的行進角度分別為90度與90度**

**====　以下兩物件的出發位置已改變, 質量比不同(5:1), 速度向量也不同; 只有半徑相同　====**
**P5 =　-1　　8**
**P6 =　0.5000　-23.0000**
**V5 =　3　　-1**
**V6 =　-1　　2**
**狀況4: 兩物件分別以速度向量(3 -1)與(-1 2)的大小與方向移動**
**-------　發生碰撞　-------**
**物件7及8的行進角度分別為1.205147e+000度與– 1.026519e+002度(後者為– 102.6519度)**

### 3.2 討論

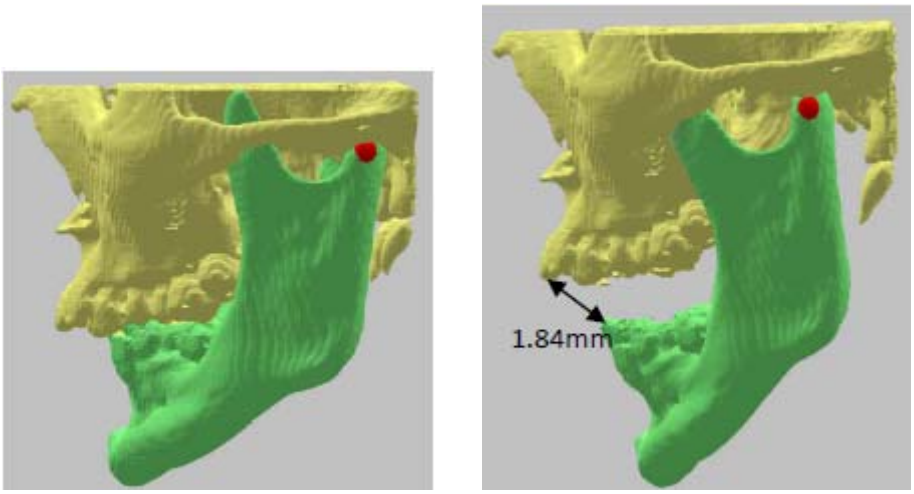
3.2.1 在碰撞偵測時，匣面範圍框比球面範圍框更適於代表複雜或不規則的物件：因為匣面範圍框有八個頂點供作前/後表面、左/右表面、上/下表面的碰撞偵測，例如一物件是否以其底面(即下表面)或側面(又分左/右表面)等登陸(landing)(接觸)另一物件，或是下降到某一水位(level)等等。但因匣面範圍框需要更多計算值，所以也會增加處理時間。
球面範圍框以其需要較少的計算值，因而易於應用且能較快速反應碰撞的發生；然而除非速度是需要被考慮的選項，否則(一般而言)以其代表繁複的不規則物件作碰撞偵測，恐易失真。

3.2.2 如同AS2.0的碰撞偵測由原本的hitTest函式，進階並細分成更精準的AS3.0之hitTestPoint函式與hitTestObject函式般，探索並開發按照實際形體或更接近形體輪廓的碰撞偵測，將會是一大挑戰且能俾益遊戲世界。

3.2.3 至於物件與圖層(level graphics)的碰撞問題，可以光跡對三角形(ray to triangle)相交與否的射影幾何檢測得到更好的碰撞偵測成效，只是較耗費時間。已有一些方法可減輕此法的時間負荷，因而有助於維持穩定的畫面速率(frame rate)；只是不在本文範圍，故不贅述 (請自行參閱相關資訊)。

3.2.4 滑鼠與物件是否發生接觸，已呈現在(3)操作。VR與AR是另外的兩大主題，不在本文範圍，請另行參閱相關資訊。附上圖16為虛擬實境的偵測張口時，顴骨與冠狀突是否碰撞的畫面。

3.2.5 實作困難處~即碰撞角度的計算與碰撞後的移動方向、碰撞後的移動速度，在"貳、方法與實作"的二與三已述及 (主要利用四維向量運算與齊次座標的射影變換) 並已實作在程式內，請參照"參、結果與討論"的(1)移動路徑與速度變慢的動畫模擬、(2)碰撞後的行進角度。惟elastic object的瞬間變形樣貌不在本研究範圍，建議參閱：碰撞的衝量原理、動量守恆原理。




圖十六　模擬下顎張口(右圖、未碰撞)、閉合(左圖)運動的碰撞偵測
(版權屬於國立成功大學、機械工程學系之虛擬實境與多媒體研究室)

## 肆、結論

一般電腦動畫採行不規則或繁複物件的矩形範圍框(bounding shape)或圓形範圍框作碰撞的測試依據，文中先行探索球面體(sphere)或匣面體(box)與無限平面(infinite plane)之間的碰撞問題，接著探討更實際的情形—即球面與任意環境的碰撞偵測；藉由導入齊次座標與四維向量，再借助射影幾何以解得碰撞發生時以及碰撞點的向量通解。最後以 MATLAB 程式語言實作移動剛體(rigid)的碰撞測試，並以給定的直線及曲線模擬移動補間動畫。實作困難處在於碰撞角度的計算與碰撞後的移動方向，甚至碰撞後的移動速度或 elastic object 的瞬間變形樣貌等。

判別兩物件的碰撞測試不但已應用於文中所述的動畫、遊戲場域，亦已應用於虛擬實境(virtual reality, VR)或擴增實境(augmented reality, AR)的使用者(user)或滑鼠與物件是否發生接觸的依據；更是網路 3D 瀏覽器判別接觸的利器，以禁制瀏覽人穿透(walking through)物件的不真行為，值得投注心力探索並開發按照實際形體或更接近形體輪廓的碰撞偵測。

## 參考文獻

[1] JA Bowers, GF Deane, RA Hyde et al., “Systems and methods for cooperative collision detection”, US Patent US9558667B2, 2017

[2] Maxim Tatarchenko, Alexey Dosovitskiy, Thomas Brox; “Octree Generating Networks: Efficient Convolutional Architectures for High-Resolution 3D Outputs”, Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2088-2096T

[3] Xuhui Fan, Bin Li, Scott Sisson; “The Binary Space Partitioning-Tree Process”, Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, PMLR 84:1859-1867, 2018

[4] SJ Friston, A Steed, “Real-Time Collision Detection for Deformable Characters with Radial Fields”, IEEE transactions on visualization and Computer Graphics, Volume: 25, Issue: 8, Aug. 1 2018

[5] Christer Ericson, “Real-Time Collision Detection”, The Morgan Kaufmann Series in Interactive 3D Technology, Elsevier, 2005

[6] Dev Ramtal and AdrianDobre, “The Essential Guide to Physics for Flash Games, Animation, and Simulations”, Springer, 2011.

[7] YJ Kim, JH Woo, MS Kim, G Elber, “Interactive tree modeling and with collision detection and avoidance”, Computer Animation and Virtual Worlds, Volume26, Issue3-4, May-August 2015

[8] JS Park, C Park, D Manocha, “Efficient probabilistic collision detection for non-convex shapes”, IEEE International Conference on Robotics and Automation, 2017

[9] D Schneider, E Schömer, N Wolpert, “Collision detection for 3D rigid body motion planning with

narrow passages”, IEEE International Conference on Robotics and Automation, 2017

[10] U Schwesinger, R Siegwart et al., “Fast collision detection through bounding volume hierarchies in workspace- time space for sampling-based motion planners”, IEEE International Conference on Robotics and Automation, 2015

# 附錄（程式）

**程式 1**

```
function anim_tween
clear, clc
% Vertices
x(1)=-0.1; y(1)=-1;
x(2)=20; y(2)=-1;
x(3)=20; y(3)=-1.3;
x(4)=-0.1; y(4)=-1.3;

% Draw patch
hold on
figure(1);
patch(x,y,'y');
% Axes settings
legend('障礙物',4)
ylabel('距離', 'FontSize',14);
set(gca,'FontSize',14);
grid on;

vo = 50;
g = 9.82;
dt = .01;
dT = dt;
t(1) = 0;
y(1) = 25;
for i = 2:2000
      t(i) = t(i-1)+dt;
      dy = vo*dt - g*t(i)*dT;
      y(i) = y(i-1)-dy;
      if y(i) < 0
            y(i) = 0;
            vo = (-1/2)*vo;
            dT = (-6/5)*dt;
      end
 end

% Draw initial figure
title('Rigid ball object collides an obstacle in 2D space.')
set(gcf,'Renderer','OpenGL');
h = plot(0,y(1),'o','MarkerSize',20,'MarkerFaceColor','b');
set(h,'EraseMode','normal');
xlim([-1,15]);
ylim([-5,30]);

% Animation Loop
i = 1;
while i<=length(y)
      set(h,'XData',t(i),'YData',y(i));

      %drawnow;
      i = i+1;
end
plot(t,y)
gtext('\leftarrow 軌跡','FontSize',14)
```

**程式 2**

```
function anim_simu
clear, clc
% Vertices
x(1)=-0.1; y(1)=-1;
x(2)=20; y(2)=-1;
x(3)=20; y(3)=-1.3;
x(4)=-0.1; y(4)=-1.3;

% Draw patch
hold on
figure(1);
patch(x,y,'y');
% Axes settings
set(gca,'FontSize',14);
grid on;

vo = 50;
g = 9.82;
dt = .01;
dT = dt;
t(1) = 0;
y(1) = 25;
u = 0;
for i = 2:2000
      t(i) = t(i-1)+dt;
      dy = vo*dt - g*t(i)*dT;
      y(i) = y(i-1)-dy;
      if y(i) < 0
            u = u + 1;
            y(i) = 0;
            vo = (-1/2)*vo;
            dT = (-6/5)*dt;
            if u<=2
             fprintf('第%i次碰撞地面; 第3次碰撞地面後,
```

```
            就只在地面移動了\n',u);
            end
        end
end

% Draw initial figure
plot(t,y)
set(gcf,'Renderer','OpenGL');
h = plot(0,y(1),'o','MarkerSize',20,'MarkerFaceColor','b');
set(h,'EraseMode','normal');
xlim([-1,15]);
ylim([-5,30]);

% Animation Loop
i = 1;
while i<=length(y)
    set(h,'XData',t(i),'YData',y(i));
    drawnow;
    i = i+1;
end
```

**程式 3**

```
function anim_fall
clear, clc
% Vertices
x(1)=-0.1; y(1)=-1;
x(2)=20; y(2)=-1;
x(3)=20; y(3)=-1.3;
x(4)=-0.1; y(4)=-1.3;

% Draw patch
hold on
figure(1);
patch(x,y,'y');

% Axes settings
set(gca,'FontSize',14);
grid on;

vo = 50;
g = 9.82;
dt = .01;
dT = dt;
t(1) = 0;
y(1) = 25;
for i = 2:2000
    t(i) = t(i-1)+dt;
    dy = vo*dt - g*t(i)*dT;
    y(i) = y(i-1)-dy;
    if y(i) < 0
        y(i) = 0;
        vo = (-1/2)*vo;
        dT = (-6/5)*dt;
    end
end

% Draw initial figure
set(gcf,'Renderer','OpenGL');
h = plot(0,y(1),'o','MarkerSize',20,'MarkerFaceColor','b');
set(h,'EraseMode','normal');
xlim([-1,15]);
ylim([-5,30]);

% Animation Loop
i = 1;
while i<=length(y)
    set(h,'XData',t(i),'YData',y(i));
    drawnow;
    i = i+1;
end
plot(t,y)
```

**程式 4**

```
function anim_angle
clear;
clc;

% objects' radius
r=1
% objects have the same unitary mass
MassCouple1=[5 1]
k = 1;
P1=[0 0]     % position data
P2=[1 0]
fprintf('=======  以下兩物件的出發位置分別固定且半徑相同  =======\n');
V1=[1 1]         % direction in which object goes
V2=[-1 1]
fprintf('狀況%i: 質量5:1，兩物件分別以速度向量(1 1)與(-1 1)各自朝右上45度/左上45度移動\n',k);
[theta1,theta2]=rigidCollision(P1,P2,V1,V2,MassCouple1,r,k);
```

```
k = k+1;
MassCouple2=[1 1]
fprintf('狀況%i: 質量比已改變成1:1，速度向量與狀況%i相同\n',k,k-1);
[theta1,theta2]=rigidCollision(P1,P2,V1,V2,MassCouple2,r,k);

k = k+1;
V3=[0 1]
V4=[0 1]
fprintf('狀況%i: 質量比與狀況%i相同(即1:1)，但兩物件分別以速度向量(0 1)與(0 1)(皆)向上移動\n',k,k-1);
[theta1,theta2]=rigidCollision(P1,P2,V3,V4,MassCouple2,r,k);

fprintf('\n=======  以下兩物件的出發位置已改變, 質量比不同(5:1), 速度向量也不同; 只有半徑相同 =======\n');
k = k+1;
P5=[-1 8]    % position data
P6=[1/2 -23]
V5=[3 -1]
V6=[-1 2]
fprintf('狀況%i: 兩物件分別以速度向量(3 -1)與(-1 2)的大小與方向移動\n',k);
[theta1,theta2]=rigidCollision(P5,P6,V5,V6,MassCouple1,r,k);
function [theta1,theta2]=rigidCollision(pos1,pos2,V1,V2,M,r,k)
% the mass difference between the objects
m21=M(2)/M(1);
% position difference
x21=pos2(1)-pos1(1);
y21=pos2(2)-pos1(2);
vx2=V2(1);
vy2=V2(2);
vx1=V1(1);
vy1=V1(2);

% velocity difference
vx21=vx2-vx1;
vy21=vy2-vy1;

% objects masses may be different
m1=M(1);
m2=M(2);

% computes the common velocity
vx_cm = (m1*V1(1)+m2*V2(1))/(m1+m2) ;
vy_cm = (m1*V1(2)+m2*V2(2))/(m1+m2) ;

% check if they are really colliding or not
if ( (vx21*x21 + vy21*y21) >= 0)
      theta1=atan2(V1(2),V1(1));
      theta2=atan2(V2(2),V2(1));
      fprintf('\n-------  沒有發生碰撞  -------\n')
     fprintf('物件%i及%i的行進角度分別為%i度與%i度\n',2*k-1,2*k, rad2deg(theta1),rad2deg(theta2));
      return;
end
% finite precision approximation
fy21=1.0E-12*abs(y21);
if ( abs(x21)<fy21 )
      if (x21<0)
         sign= -1;
      else
         sign=1;
      end
      x21=fy21*sign;
end

% update velocities
a=y21/x21;
dvx2= -2*(vx21 +a*vy21)/((1+a*a)*(1+m21)) ;
vx2=vx2+dvx2;
vy2=vy2+a*dvx2;
vx1=vx1-m21*dvx2;
vy1=vy1-a*m21*dvx2;

% velocity correction for inelastic collisions
vx1=(vx1-vx_cm)*r + vx_cm;
vy1=(vy1-vy_cm)*r + vy_cm;
vx2=(vx2-vx_cm)*r + vx_cm;
vy2=(vy2-vy_cm)*r + vy_cm;

% transform the exit angles of the objects
fprintf('\n-------  發生碰撞-------\n')
theta1=atan2(vy1,vx1);
theta2=atan2(vy2,vx2);
fprintf('物件%i及%i的行進角度分別為%i度與%i度\n',2*k-1,2*k,rad2deg(theta1),rad2deg(theta2));
```

# Implementing Collision Detection with Tweened Animation

Shing-Min Hu

## Abstract

Collision detection is not only applied in computer animation and computer game but also in virtual reality (VR) and augmented reality (AR) to test if users or users' mice contact objects, furthermore to prohibit browser's walking through objects incorrectly in 3D Web Browsers. Bounding shapes are served as the criterion for collision detecting. Common practices of bounding shapes are rectangles and circles for 2D graphics, and boxes and spheres for 3D graphics engines. The paper explores the calculations involved in determining when a sphere or box collides with an infinite plane. Then the article discusses the more difficult, but very practical method of resolving the collision of a sphere with an arbitrary environment. Finally the study employs MATLAB to model rigid sphere in motion colliding with ground, and to model tweened animation by setting straight lines and curves. The exercise also takes the gravity into account and is programmed to speed up when falling or to slow down when bouncing back. What the implementation is hard is to compute collision angles and the velocity after colliding, let alone elastic objects which are not examined here.

Keywords: Collision Detection, Projective Transformation, In-between Animation

ottohu@mail.tf.edu.tw
Department of Digital Game and Animation Design, Tung-Fung Design University, Kaohsiung, 82941, Taiwan, ROC