第十八章

槍枝與爆炸

在電玩遊戲中其中一種相當流行的類型是第一人稱的射擊遊戲。自從德軍總部 *3D* 和毀滅戰士出現後，就吸引許多人投入研發經費到這個領域內。但讓人感到驚訝的是槍枝的瞄準和子彈在空氣中飛行的物理行為都很少能正確被模擬出來。一般來說，遊戲開發者都將槍枝視為雷射光束，因此子彈都是朝著無限延伸的直線去飛行。我們在本章將要討論的是，如何更精確去模擬子彈的瞄準和飛行軌跡（稱為彈道學）。

拋體運動

彈道學可分成四個主題來討論。「內部彈道學」是研究子彈在槍管內的行為；「轉移彈道學」是研究子彈要離開槍管時的行為。一旦子彈完全離開了槍管，就是「外部彈道學」的範疇了。此時唯一的加速度即是重力，而受到的外力影響則與第六章談到的相同。最後是「終端彈道學」，與子彈擊中目標後的行為有關。本章會討論後兩個主題，因為與射擊者較有關，而前兩個主題則與槍支製造商較有關。若忘記第六章的內容，建議再繼續閱讀本章前先複習一次。

雖然可以不理會槍管內子彈的行為，但仍有些事情是需要弄清楚的。第一就是槍管指向何處，亦即槍的目標為何，其通常是由螢幕上滑鼠位置來控制的。

再來是子彈的初速度。此處的「子彈」是指實際上離開槍管的金屬彈殼。至於射擊前裝填到槍裡的則包含彈殼、火藥、引信以及子彈。通常是在子彈離開槍口（槍管末端）後才去計算初速度，因此可稱為**槍口速度**。各種彈藥在工廠生產時都會經過測試，並有固定的槍口速度。若在遊戲中要增加擬真度的話，可以賦予不同彈藥不同的槍口速度。如此一來，手槍子彈和來福槍子彈的各自範圍就不會相同了。各種彈藥的子彈也會有一定重量，以公克或喱為單位。1 喱等於 0.0648 公克，其值是由小麥單粒種子的重量所得出的。

最後還要考慮空氣阻力對子彈飛行的影響，這也是彈道學有趣之處。但此處不會涉及到太過複雜的氣體動力學，而是使用現成的阻力模型。接著要來檢視目前第一人稱射擊遊戲的物理問題。

對遊戲開發者來說，要在遊戲中使用槍械會有一些特殊問題。若你曾經去過靶場實際打靶過，就會知道要準確射中目標需要良好的練習和專注力。射擊的難度相當高，使其在嚴格控制的環境下能成為一項奧運的運動，所以相比之下，遊戲中的角色能從掩蔽處突然跳起來，並且使用五顆子彈射中五個敵人簡直是超人般的行為。射擊遊戲中要打中遠方的物體，玩家只需要將十字準線對準目標再按下滑鼠按鍵。事實上，要用幾公克的子彈要打中幾百公尺外的物體，所需的技術非常複雜，所以若有人能夠很穩定的射中目標是一件讓人很驚奇的事。所以對開發者來說，在遊戲中是否要真實模擬出槍枝性能，是值得考慮的一件事。

要歸結出子彈飛行時發生的物理行為並不容易，但這對實際的射擊卻是非常重要的。已經有許多方法可用來比較各式彈藥，以便讓人能預測它們性能上的優劣，**彈道係數**（*ballistic coefficient*，*BC*）就是其中一種方式。彈道係數是特定子彈跟某個標準子彈相比下能夠維持其發射速度的比率值。常見的標準是 G1 系列，然而其使用上仍有限制，因為並未考慮現今子彈的形狀能降低許多阻力。後續有一些新的計算方式，例如 G2、G3 和 ECT。若對於如何建構高度準確的彈道模型有興趣的話，有一些免費的程式可供深入參考，像是 Remington's Shoot! 以及 GNU Ballistics 程式。多數第一人稱的射擊遊戲並不考慮風力影響和彈道垂落線，而此處僅使用第六章拋體演算法中簡化的做法，去調整已存在的參數。

瞄準

當談到槍枝瞄準的問題時，主要類型是以來福槍或卡賓槍為主。手槍通常不用於遠距離射擊，霰彈槍一般則不會特別瞄準，而是直接射擊，因此都不在討論範圍內。這兩種武器稱為「直射武器」。稍後會討論到來福槍的直射距離，但對於手槍和霰彈槍來說，在有效射程距離內可以合理地預期能射中指向的目標。但這不代表它們不需要事先瞄準，但在第一人稱射擊的快速戰鬥遊戲中，若還要玩家花時間瞄準則顯得不太理想。因此大部分遊戲都是採用「直接掏槍出來射擊」或「自由射擊模式」，其中槍枝可能甚至和鏡頭不在一直線。細心的程式設計者在判定是否擊中目標前，會檢查目標是否在有效的射擊範圍內，即使採用自動瞄準仍是如此。有效的射擊範圍就是子彈落地前能夠行進的距離，其計算方式很明顯可參考第二章和第六章的公式。

為了要討論槍枝，此處修改了第六章的 Cannon2 範例，以便在稱為 Marksman 的 Java 程式中執行。在此處範例中，玩家會瀏覽範圍是一公尺乘一公尺的目標。因為提供的是可調整的鏡頭，因此當範圍增加時仍看得見目標。瞄準點是以一個空心圓來表示，彈孔則是以黑色實心圓來表示。用來決定模擬程式中使用者槍枝瞄準位置的方法能將以像素為基礎的滑鼠位置轉為相對應的座標。此距離和範圍是用於找出槍枝瞄準所需的角度。相關程式碼如下，其中 `alp` 和 `gmm` 分別是傾斜角和軸承角。這些都是由水平視線來衡量的：

```
alp = 90-Math.toDegrees(Math.atan(((200-aimY)*(targetH/(drawH)))/range));
gmm = Math.toDegrees(Math.atan(((200-aimX)*(targetH/(drawH)))/range));
```

其中，`targetH/drawH` 是目標高度和螢幕上以像素表示的目標高度的比例。如此可將滑鼠以像素表示的座標轉成以公尺為單位。反正切函數再將這些距離的比例轉成槍枝需要的角度。常數 200 是指在像素座標系統中，與目標的距離是 200 像素。若要應用在 3D 繪圖系統，可以移除此處所需做的一些轉換動作。

在圖 18-1 所示的 10 公尺範圍內，算是近距離的射程，且彈孔與瞄準器方向一致。

圖 18-1：近距離的射程

圖 18-2 中，目標位於 100 公尺處，因為某些原因的關係，子彈並未擊中所瞄準的目標。在 300 公尺時，仍然沒有射中目標。你可以注意到，納入簡單的投影運動和理想阻力的計算後，已經遇到如何瞄準目標中心的問題了。處理這類問題的過程就稱為「調整槍械瞄準具」（*zero the sights*）。

圖 18-2：擊中位置低於預期目標

調整槍械的瞄準具

若想讓遊戲中的槍戰更為逼真，如何模擬調整槍械的瞄準具就顯得非常重要。如之前所述，若玩家需要在室內不斷移動，可能不會想去考慮風力和距離的問題。然而若是模擬狩獵或狙擊的遊戲，就很適合將此納入考量。

當一個人使用瞄準鏡時，其身體和來福槍變成一個剛體，因此要改變武器的瞄準方向，必須轉動整個身體。因為玩家通常透過左手去使用鍵盤來控制射擊者位置，右手則是控制射擊方向，對於我們來說相當便於設計。由於其他瞄準方式，像是前面提到的自由射擊模式，並不存在於現實世界中，所以此處會集中討論的是剛體的射擊。

彈道垂落線：重力和空氣阻力

若以水平方式使用來福槍瞄準，則可以預期子彈是以水平離開槍口的，而重力和空氣阻力會使其落下。圖 18-3 顯示的是來福槍和瞄準具平行放置的組合。先暫時忽略其他因素，此處子彈明顯不會擊中所瞄準的位置，而是低於幾公分。顯示的是來福槍和瞄

準具平行放置的組合。先暫時忽略其他因素，此處子彈明顯不會擊中所瞄準的位置，而是低於幾公分。

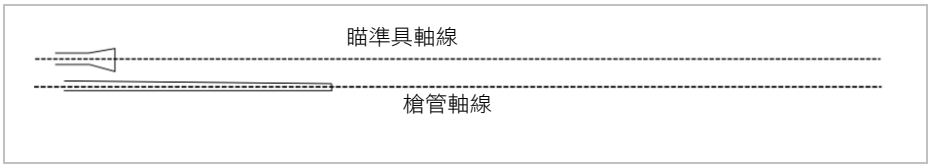

圖 18-3：水平瞄準具

接著調整瞄準具在垂直方向上的高低，可以讓來福槍準確打中瞄準具指向的目標（參考圖 18-4）。子彈越過瞄準具所定義的瞄準線的距離，稱為「零點距離」。若目標位於零點距離內，只需要將十字準線對準目標並且扣下扳機。

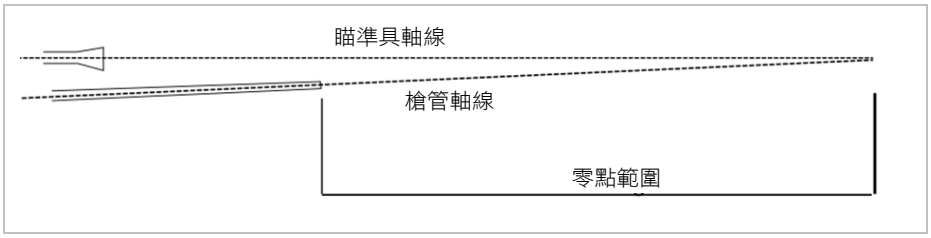

圖 18-4：瞄準具的垂直高低

若移除目標，且畫出彈道軌跡，就如圖 18-5 所示。

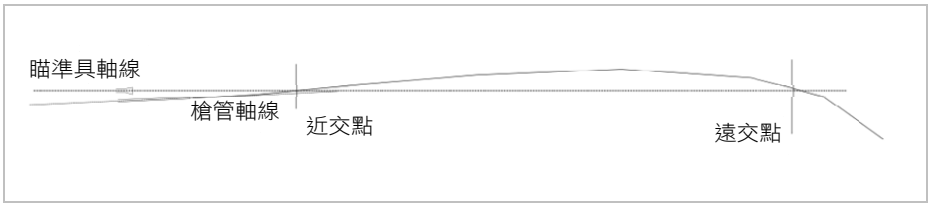

圖 18-5：彈道

其中可以看到彈道跟瞄準線有第二個交點，稱為遠交點（*far zero*），而這通常就是瞄準點的位置。這樣的圖表通常是由彈藥供應商所提供的。更多的時候，他們會提供縱向射程高度的彈道表。當使用這些圖表時，必須注意到它們會假設瞄準具是水平的，且子彈一開始位於低於瞄準線的位置（高度為負值）。表 18-1 顯示的資料來自

Remington 網站，並且假設將來福槍的瞄準具調整在 100 碼。由此表格可看出每 50 碼時，子彈低於水平線多遠。若要符合表格的情況，可以透過在拋體模擬器中調整阻力係數來達成。

表 18-1：長程彈道：來福槍子彈 Remington Express .45 – 70 Govt

範圍（碼）	100	150	200	250	300	400	500
垂落線（英寸）	0	– 4.6	– 13.8	– 28.6	– 50.1	– 115.47	– 219.1

從此表格可以發現彈道垂落線在 250 碼時超過 2 英吋。這意謂著若目標為在 250 碼之處，而瞄準具對準的是 100 碼，則瞄準點必須在目標物上方 2 英吋之處才能擊中目標。如此調整高度勢必造成無法透過瞄準具看到目標物。

要克服此問題，大部分來福槍的瞄準具側邊都有旋鈕可供垂直方向的調整。多數瞄準具每次轉動所調整的大小是 1/4 MOA（minute of angle），但有些則是 1/8、1/2 或 1 個 MOA。MOA 是角度上一「分」的意思，亦即一度的 1/60。因此當調整垂直高度時，射擊者可以調整瞄準具旁的旋鈕，然後只要聽旋鈕在不同刻度之間轉動次數，即可知道轉了多少 MOA。當重新將十字準線對準目標時，槍管的角度會略有不同，基本上是將瞄準位置調高以適應較遠距離的目標。

若要提高射擊的精準度，射擊者需要調整來福槍瞄準位置到某個範圍。然後當嘗試射擊時，再預估到目標的範圍並適當調整瞄準具。而發生錯誤的最大原因在於對距離的估算不夠準確。現代的射擊者通常會使用雷射測距儀來精確決判定需要的高低偏差值為何。在長距離射擊的情況，可以提供範圍資訊給使用者，並讓他們能將來福槍的瞄準具由目前的零點範圍調整到一個新的零點範圍。

現今大多數的遊戲甚至沒有模擬重力對子彈的影響，因此在狙擊或狩獵的遊戲中，加上這樣的垂直方向調整可以讓遊戲增加所需的準確性。

風力

如同第六章的拋體，此處射擊遊戲的子彈也會受到風力的影響，而影響程度取決於側向阻力係數。在模擬中，可以透過調整因子 `Cw` 來改變此影響程度。同樣地，這只會作用於來福槍長距離射擊時。在 20 公尺，風力對子彈沒有任何影響。在 600 公尺，則會讓子彈有一公尺的偏差。對風力的調整如同垂直方向的調整一般。藉由轉動旋鈕，可

以將瞄準具稍微由槍管中心線往左或往右移動。這個角度在此稱為 γ，可用來取消風力所產生的影響。

要處理風力的問題，需要依照射擊者所用彈藥的特定效能來做一些簡單計算。若射擊時有風力經過，則幾英吋的調整可能就帶來一半風速的影響（英里／小時）。當然，對於每個彈藥口徑來說，會有不同的經驗法則，但在此處的模擬中，可以任意將風力阻力係數調整成自已希望的值。如此一來，遊戲中的角色必須像真實世界中的射擊者一般，花些時間弄清楚風力對其射擊的影響程度。當風力改變時，必須立即調整對目標物的瞄準位置或改變瞄準具的風阻設定。

呼吸與身體姿勢

雖然大部分遊戲在計算子彈軌跡時都不會考慮重力和風力，但許多遊戲仍試圖調整子彈發射的準確性。最常見的是遊戲開發者使用四線不相交的十字準線。當射擊出去時，子彈會落在由此四條線的內部端點構成的圓形範圍內。不同的武器會有不同的準確性，因此可以適度去調整每條線。通常來說，第一次射擊會最準確，而一旦射擊後，必須重新瞄準目標，這會花費一些時間。因此，若連續射擊的話，準確性會愈來愈低。

此處有模擬一些會影響到真實世界準確性的事情，對於其他可能涵蓋的因素也會提供一些建議。因為將遊戲的目標放在來福槍的遠距離射擊，所以造成誤差的來源就是射擊者的呼吸。如之前所討論，當射擊者透過來福槍的瞄準鏡觀測時，他就固定不動。當他呼吸，來福槍也會跟著有所起伏。在嘗試射擊相當困難的目標時，射擊者通常都會先喘口氣再將槍枝握緊。此處的模擬中，使用一個 `breathing` 類別來處理瞄準點受射擊者呼吸而上下起伏的情況。在函式 `initComponents()` 中有個計時器，會每 100 毫秒發射一次。而從程式碼可知，這會造成每兩秒會有一次呼吸。

```
timer = new Timer(100, TargetPanel);
timer.start();
```

該函式依據下列演算法能讓瞄準點 (`aimX`, `aimY`) 的移動不受滑鼠游標影響：

```
if (direction == true) {
    breathHeight = breathHeight + 1;
    if (breathHeight == 5) {
```

```
            direction = false;
            breathHeight = breathHeight + 1;
        }
    }

    if (direction == false) {
        breathHeight = breathHeight - 1;
        if (breathHeight == -5) {
            direction = true;
        }
    }

    if (breathing) {
        aimY = aimY + breathHeight;
    }
```

程式中簡單地讓來福槍往上移動到某個限定值，此處是五個像素，然後再往下移動。更好的實作方式則是在使用者將鏡頭放大時，搖晃程度也放大，因此呈現出瞄準點不穩定的情況。至於要用什麼創新功能將瞄準圈遠離滑鼠游標，以模擬真實的槍枝瞄準則不受限。然而因為是第一人稱射擊遊戲，所以可變化的類型較少。

當準備射擊時，玩家可以點擊滑鼠左鍵來屏住呼吸，變數 `breathing` 是 `false`，十字準線會停止移動。這樣的設計可以使遊戲更具挑戰性和吸引力。另外，要注意到在現實中，當射擊者屏住呼吸太久因為身體缺氧的關係，瞄準點又會變得不穩定。因此程式中可以設計成當已經按下滑鼠左鍵一段時間後，就讓瞄準點變得不穩定。

許多遊戲也根據人體姿勢來調整武器的精準度。基本的射擊姿勢有三種，站立、跪著和俯伏。其中，跪下是整種形式的蹲姿，而非僅是膝蓋跪著。俯伏則是人全身貼著地平面。因為來福槍使用時會貼著人體，所以身體姿勢愈穩定，瞄準點就愈穩定。當站立時，身體肌肉必須出許多力維持站姿。當跪著時，則出力較少。而當俯伏時，則完全不用出力。

你可以在每次瞄準時加入這些參數，並做調整來改變每個姿勢的相對優勢。不過，俯伏會比跪著來得穩定，跪著又比站立更加穩定。

後座力與衝擊

討論過子彈的瞄準、射擊和位置後，現在可以來討論最後一階段：終端彈道。若要完全了解子彈最後飛行的情況，就要從頭開始談起。稍早談到反彈是由於牛頓的動量守恆定律。大家應該都看過電影中一個老套的情節：一個英雄開槍射擊壞人，子彈把壞人給擊倒。但這其實大有問題，因為若子彈射出的力道大到足以擊倒一個人，則射擊者也會被槍枝的後坐力給擊倒。因為事實上兩者感受到的力道應該幾乎是相同的。若一個 9 毫米子彈，重量 7.45 公克，以 390 公尺/秒的速度離開槍管，則槍枝產生的後座力，使得它的動量等於子彈的力矩。

遊戲中的太空場景引進後座力會很有趣。在地球上，槍枝的後座力很快因為摩擦力的關係而傳遞到人與地球之間的地平面。在太空中，射擊者沒有星體來產生阻力，所以槍枝的後座力變成了槍枝與人系統中的後座力。下一次若想讓遊戲中的角色在微重力環境下於太空船之間移動，可以讓其藉由發射子彈的後座力來達成。

此時若一個人被擊中，很快就會倒下來，但整個過程會跟生物學比較相關。然而，先忽略活生生的目標，若要模擬子彈擊中某個物體造成的傷害，很重要的是要考慮子彈的動能。事實上，子彈和砲彈都是動能武器，因為它們摧毀目標的方式都是將其動能轉移到目標上。這與炸彈在衝擊後將化學能轉成熱能和動能有所不同。

爆炸

若要準確模擬爆炸場景需要考慮多物理的流體模擬，類似在第十四章到十六章我們所討論的內容。一般電玩遊戲中，通常會有一推桶子放在一起，然後只要一被射中就會產生劇烈爆炸，炸毀周遭車輛，同時也能有效對抗許多敵人。但這在實際生活中並不容易發生，例如使用手槍擊中瓦斯桶幾乎不可能著火，更別說是爆炸了。甚至用來福槍擊中丙烷罐也不至於產生火花。倘若是以 1/4 丙烷和 3/4 氧氣混合而成的桶子被擊中的話就有可能產生爆炸，但這樣的物品並不常見。無論如何，在電玩遊戲中偶而都有紅色桶子可以射擊，因此接下來要檢視如何讓遊戲中的爆炸場景可以更加真實。

粒子爆炸

對於大多數遊戲中的爆炸場景來說，使用第二章提及的粒子爆炸來模擬就很足夠了。在第二章的粒子都只是一些點，但其實粒子的形狀並不受限於此。在某些情況，例如子彈射中金屬容器而爆炸，使用粒子爆炸來模擬就很適合。然而若將粒子視為汽車的某個部分，則可以模擬汽車爆炸的情況。這很容易是因為粒子爆炸沒有任何的角運動。儘管車子被炸飛時，可以將車子不同部分當成不同粒子來看，但它們不會旋轉。好消息則是，粒子爆炸其實就足夠在遊戲中模擬爆炸的場景了。

若要詳細討論子彈和粒子爆炸間的關係，會考慮的情況比子彈炸翻一台車來得更為真實。現在考慮當一個子彈擊中一些鬆散的礫石，在碰撞時會讓礫石拋向空中。此處不會試圖計算衝擊時複雜的碰撞問題，而是以第二章的程式為基礎開發粒子爆炸的程式：

```
void CreateParticleExplosion(int x, int y, int Vinit, int life,
                             float gravity, float angle)
{
     int   i;
     int   m;
     float f;

     Explosion.Active = TRUE;
     Explosion.x = x;
     Explosion.y = y;
     Explosion.V0 = Vinit;

     for(i=0; i<_MAXPARTICLES; i++)
     {
          Explosion.p[i].x = 0;
          Explosion.p[i].y = 0;
          Explosion.p[i].vi = tb_Rnd(Vinit/2, Vinit);

          if(angle < 999)
          {
               if(tb_Rnd(0,1) == 0)
                    m = -1;
               else
                    m = 1;
               Explosion.p[i].angle = -angle + m * tb_Rnd(0,10);
          } else
               Explosion.p[i].angle = tb_Rnd(0,360);

          f = (float) tb_Rnd(80, 100) / 100.0f;
```

```
            Explosion.p[i].life = tb_Round(life * f);
            Explosion.p[i].r = 255;//tb_Rnd(225, 255);
            Explosion.p[i].g = 255;//tb_Rnd(85, 115);
            Explosion.p[i].b = 255;//tb_Rnd(15, 45);
            Explosion.p[i].time = 0;
            Explosion.p[i].Active = TRUE;
            Explosion.p[i].gravity = gravity;
        }

    }
```

如所見一般，初速度 `V0` 控制著爆炸的強度。在第二章時，這個值是隨機選取。但現在可以根據子彈在空中飛行的情況來預估產生的爆炸強度為如何。如前述所提，一個子彈在飛行時的任何時間點 `t` 都帶有能量。此能量是其動能，且等於子彈質量的一半乘以速度平方。

在拋體模擬中，當子彈在空中飛行時，計算此能量很簡單。但要注意的是，一個速度慢的大子彈其威力等同於速度快的小子彈。接下來的程式會假設動能可以百分之百傳遞給目標，但子彈不能是直接飛行穿透某物體。可以想像有兩個目標都懸掛在天花板上，分別由紙和鋼鐵做成的。當被擊中時，後者會搖晃，但前者仍保持靜止不動。這是因為子彈要直接穿透紙張，其動能無法傳遞給目標物。為了簡化問題，此處假設所有子彈的動能都能傳遞到礫石。公式如下所示：

$$1/2\ m_b\ v^2_{bullet} = \Sigma 1/2\ m_g\ v^2_{gravel}$$

請注意，這是礫石速度個別位元的總和。在第二章，每個粒子都被給予隨機速度，範圍由 `Vinit/2` 到 `Vinit`。這可能導致產生出的一組粒子，其能量超過輸入的能量。為了預防此情形發生，在類別中新增一個變數：

```
typedef     struct _TParticle
{
    float           x;              // 粒子的 x 座標
    float           y;              // 粒子的 y 座標
    float           vi;             // 初速度
    float           angle;          // 初始軌道（方向）
    int             life;           // 持續時間，單位 ms
    int             r;              // 粒子色彩的紅色成分
    int             g;              // 粒子色彩的綠色成分
    int             b;              // 粒子色彩的藍色成分
    int             time;           // 記錄效果的時間
    float           gravity;        // 重力因子
```

```
    BOOL          Active;        // 指出粒子是「活的」或「死的」

    float         mass;          // 用於計算粒子的能量
} TParticle;

#define     _MAXPARTICLES 50
#define    _MASSOFPARTICLE .25

typedef struct _TParticleExplosion
{
    TParticle          p[_MAXPARTICLES];  // 粒子列表
                                          // 構成這種效果
    int           x;  // 初始的 x 座標
    int           y;  // 初始的 y 座標
    float         KE; // 可用 Kinect 的能源
    float
    BOOL          Active;      // 表示此效果是否存在

} TParticleExplosion;
```

請注意，因為爆炸強度由可用的動能決定，所以已經不再需要 V0 了。子彈的動能由變數 KEb 儲存，新的 CreateParticleExplosion 函式如下：

```
void CreateParticleExplosion(int x, int y, int KEb, int life,
                             float gravity, float angle)
{
    int     i;
    int     m;
    float   f;

    Explosion.Active = TRUE;
    Explosion.x = x;
    Explosion.y = y;
    Explosion.KE = KEb;

    for(i=0; i<_MAXPARTICLES; i++)
    {
        Explosion.p[i].x = 0;
        Explosion.p[i].y = 0;
        Explosion.p[i].m = _MASSOFPARTICLE; // 單一礫石的質量
        Explosion.p[i].vi = tb_Rnd(0, sqrt(Explosion.KE/(_MASSOFPARTICLE*
                                                    _MAXPARTICLES));
        Explosion.KE = Explosion.KE - ((1/2)*(Explosion.p[i].m)*
                                                (Explosion.p[i].vi));

        if(angle < 999)
        {
            if(tb_Rnd(0,1) == 0)
                m = -1;
            else
                m = 1;
```

```
                Explosion.p[i].angle = -angle + m * tb_Rnd(0,10);
            } else
                Explosion.p[i].angle = tb_Rnd(0,360);

            f = (float) tb_Rnd(80, 100) / 100.0f;
            Explosion.p[i].life = tb_Round(life * f);
            Explosion.p[i].r = 255;//tb_Rnd(225, 255);
            Explosion.p[i].g = 255;//tb_Rnd(85, 115);
            Explosion.p[i].b = 255;//tb_Rnd(15, 45);
            Explosion.p[i].time = 0;
            Explosion.p[i].Active = TRUE;
            Explosion.p[i].gravity = gravity;
        }

    }
```

如程式碼所示，此處會隨機設定粒子的初速度，範圍從 0 到能夠消耗爆炸時所有動能的值。下一行程式碼則將爆炸中可用的動能減掉剛分配給粒子的部分，如此可確保產生出的爆炸威力會減弱。至於更有趣的做法則是以給定的質量分佈來初始化粒子，且以常態分配的方式指定速度，而非隨機指定。詳細內容可參考《*Numerical recipes in C*》一書。

儘管前述程式碼未考慮到動能轉換的一些細節，但能確保較小型且慢速的子彈能產生出較小的爆炸效果。這也是目前電玩遊戲普遍缺乏的部分。

多邊形爆炸

物體爆炸後，若產生的是小型且形狀一致的區塊，就適合使用粒子爆炸來處理。但若爆炸後產生的是各種不規格形狀的區塊則不適用。因此電玩遊戲中鮮少看到離玩家很近的地方發生汽車爆炸或門板飛落到地上。相反地，遊戲通常以粒子爆炸來處理這類情形，因此繪圖時會呈現出已經炸裂後的碎片。

若想完整模擬剛體的爆炸場景，可以重複使用處理位移的粒子爆炸程式碼，只是粒子代表的是剛體重心。

同時需要加入初始角速度，並如同第十二章讓模擬器處理初始角度後的移動。

因為本書能收入的內容有限，無法再舉其他範例來說明，所以接著來解釋一下引起一場爆炸所需的輸入能量。請回憶到子彈本身沒有能量造成造成物體分裂，即使用坦克機槍射中某個物體，也並非子彈的動能造成物體分裂，而是擊中後的「誘爆」（secondary explosion）所引起的。以一輛坦克擊中另一輛坦克來說，衝擊產生的熔渣會佈滿後者內部，使其燃料或是彈藥爆炸。如此一來就會發生巨大的爆炸聲響，此即為化學能轉成熱能、光以及壓力。

量化武器化學能量的常見方式是使用「*TNT 當量法*」（*TNT equivalency*），意義是爆炸時所釋放的能量相當於多少噸 TNT 炸藥爆炸所釋放的能量。要模擬汽油和空氣的爆炸很複雜，所以此處以 TNT 來衡量。一公斤 TNT 含有 4.184 兆焦耳的能量；9 釐米子彈則大約是 400 焦耳。兩者相比就能知道為何子彈無法爆破東西，但 TNT 炸藥卻很容易做到。

為了能夠討論此議題，假設遊戲中的角色將 1 公斤的 TNT 炸藥扔進一個五邊形的盒子中。當 TNT 被引爆時，可以給盒子的每個邊一個初速度值（平移速度和角速度），然後再採用運動學公式。所有速度都是基於以下兩項簡單的原則：

- 速度向量可由兩點決定：TNT 炸藥的中心以及五邊形盒子的中心。
- 所有動能的和必需小於 TNT 中的化學能量。這可根據盒子和 TNT 距離的平方按比例分配。

在第一項原則中，使用多邊形區域的中心會讓多邊形產生旋轉，因為有繞著重心的力矩。但若兩者同時發生，則不會旋轉。但因為空氣動阻力以及不均勻的爆炸仍會造成旋轉，所以必須模擬此情況或是手動賦予一些旋轉速度。

既然速度有了方向，就要再來定義其大小了。因為炸藥引爆而產生的熱能，會使氣體迅速膨脹，因此對周遭的物體產生作用力。然而許多化學能量其實都會變成熱能、光和聲音。而無法轉移給物體。基本上來說，只有三分之一的化學能量在初始的爆炸中會轉換。這可稱為爆炸的效率，以符號 ζ 來表示。 多邊形速度之間的關係可以表示如下：

$$\zeta E_{chemical} = \sum_{i=0}^{3} \frac{1}{2}\left(m_i v_i^2 + I_i w_i^2\right)$$

其中，可以調整 ζ 來讓多邊形在爆炸時有更真實的速度（例如避免以光速傳輸）。若爆炸產生的能量是平均分配，則較輕的物體一如預期會有較快的速度。也可以依據物體與爆炸地點的距離，給予適當權重，來調整釋放給每個物體的爆炸能量。程式中也該調整爆炸產生的壓力，但通常來說，其會隨距離的立方而減少，並隨時間呈指數減少。

若讀者想模擬更複雜的爆炸場景，有許多不錯的資料可供參考，例如建築物如何反應炸彈爆炸，美國聯邦緊急事務管理署（FEMA）、陸軍（Army）和海軍（Navy）都有相關的論文可供參考，像是「*FEMA 426 Reference Manual to Mitigate Potential Terrorist Attacks Against Buildings*」這樣的文件，其中的觀念有助於更真實去模擬建築物在爆炸後所受的損傷。