XVIII. Broń i wybuchy

Jednym z najpopularniejszych gatunków gier wideo jest czcigodny strzelec z perspektywy pierwszej osoby. Od czasu przełomowych gier Wolfenstein 3D i Doom strzelanka jednosobowa otrzymała największą część budżetu na badania i rozwój. To zdumiewające, że fizyka celowania karabinem i kuli podróżującej w powietrzu rzadko jest dokładnie modelowana. Ogólnie rzecz biorąc, projektanci gier traktują pistolety jak wiązki laserowe, tak że wszędzie, gdzie je wskażesz, kula trafia w nieskończenie prostą linię. W tej części omówimy, jak dokładniej modelować celowanie i trajektorię pocisków, co jest znane jako balistyka.

Ruch pocisku

W rzeczywistości istnieją cztery podtematy balistyki. Balistyka wewnętrzna to badanie tego, co dzieje się z kulą wewnątrz lufy pistoletu; balistyka przejściowa jest nauką co się dzieje, gdy kula wychodzi z lufy. Gdy kula całkowicie opuści luf1), znajduje się ona w sferze zewnętrznej balistyki. W tym momencie jedynym przyspieszeniem jest grawitacja i te same siły, o których mowa w części 6, przejmują kontrolę. Ostatnim tematem jest balistyka terminalowa, czyli badanie tego, co dzieje się, gdy pocisk trafi w cel. Ostatnie dwa tematy to te, które omówimy tutaj. Pozostałe fazy są ważniejsze dla producentów broni palnej, a nie dla strzelca. Jeśli nie pamiętasz materiału z Części 6, zdecydowanie zalecamy przejrzenie go przed kontynuowaniem. Chociaż nie jesteśmy bardzo zaniepokojeni, co dzieje się w lufie pistoletu, istnieje kilka ciekawostek, które musimy wiedzieć o systemie. Pierwszy to miejsce, w którym lufa jest ostro zakończona. Nazywa się to cel pistoletu i prawie jest powszechnie kontrolowane przez to, gdzie mysz jest na ekranie. Następna jest prędkość początkowa pocisku. Pocisk odnosi się do rzeczywistego pocisku metalowego, który opuszcza lufę; to, co ładujesz do pistoletu, nazywa się rundą i zawiera osłonę, proch strzelniczy, elementarz i, oczywiście, kulę. Początkowa prędkość jest zwykle mierzona tuż po tym, jak pocisk opuści kufę (koniec lufy) i jest odpowiednio nazywany prędkością wylotową. Każda amunicja jest testowana w fabryce i otrzymuje prędkość wylotową. Możesz dodać realizmu do swojej gry, nadając amunicji różne prędkości wylotowe. W ten sposób runda pistoletowa nie będzie miała takiego samego zasięgu jak runda karabinu. Amunicja ma również masę punktową mierzoną w gramach lub ziarnach. Jedno ziarno jest równe 0,0648 gramów i jest starą jednostką opartą na wadze pojedynczego ziarna pszenicy! Wreszcie, potrzebujemy trochę przybliżenia, w jaki sposób opór powietrza wpłynie na lot kuli. Tutaj zaczynają interesować się ludzie uczący się balistyki, ale będziemy trzymać się z daleka od egzotycznej aerodynamiki i wykorzystywać nasz istniejący model oporu. Po pierwsze, powinniśmy przejrzeć aktualny stan fizyki strzelanki pierwszej osoby. Broń palna w grach stanowi wyjątkowy problem dla twórcy gry. Jeśli kiedykolwiek byłeś na strzelnicy, wiesz, że w rzeczywistości wymaga dużo praktyki i koncentracji, aby trafnie trafić w cel. Biorąc pod uwagę, że strzelanie do celu jest wystarczająco trudne, aby być sportem olimpijskim w bardzo kontrolowanych warunkach, zdolność postaci w grze do wyskakiwania z osłony i strzelania do pięciu wrogów za pomocą pięciu kul jest nieco nadludzka. Wszyscy graliśmy w gry, w których strzelasz do celu bardzo daleko, a procedura jest tak prosta, jak wskazanie krzyżyka w miejscu, w którym chcesz kula i kliknięcie przycisku myszy. W rzeczywistości umiejętność zdobycia pocisku ważącego kilka gramów, aby uderzyć w coś kilkaset metrów dalej, jest tak skomplikowana, że zdumiewające, że każdy robi to z regularnością. Dla tych deweloperów, którzy chcą faktycznie modelować wydajność broni palnej w swojej grze, należy wziąć pod uwagę obosieczny miecz. Fizyka tego, co dzieje się z kulą w locie, nie jest łatwa do sprowadzenia. Jednak zachowanie kuli podczas jej przemieszczania się jest ważne w praktyce sztuce strzeleckiej. Wykonano wiele pracy, aby znaleźć sposób na porównanie amunicji, aby łowca lub strzelec mógł przewidzieć działanie danej amunicji. Wynikiem jest czynnik pseudofizyczny zwany współczynnikiem balistycznym (BC). BC jest współczynnikiem, który określa zdolność konkretnej kuli do zachowania jej niższej prędkości w porównaniu z jakąś standardową kulą. Najczęściej spotykaną

formą jest pocisk odniesienia G1. Jednak liczba ta ma ograniczone zastosowanie, ponieważ nie uwzględnia nowoczesnych kształtów pocisków, które zapewniają bardzo mały opór. Dostępne są zaktualizowane modele, których oznaczenia to G2, G3 i ECT. Jeśli interesują Cię szczegóły bardzo dokładnego modelowania balistycznego, istnieje kilka darmowych programów, które mogą dostarczyć Ci dogłębne modele, takie jak Remington's Shoot! oraz program balistyczny GNU. Biorąc pod uwagę, że większość strzelanek pierwszoosobowych nie uwzględnia jeszcze efektów wiatrów lub upadku pocisku, ograniczymy się do uproszczonej metody zamiany istniejących parametrów w algorytmie pocisku.

Celowanie

Omawiając cel, będziemy przede wszystkim mówić o karabinach lub karabinach, ponieważ pistolety nie są zwykle używane do strzelania na duże odległości. Podobnie, strzelby, biorąc pod uwagę, że wystrzeliwują wiele małych pocisków, generalnie nie są celowo wycelowane, ale zamiast tego wskazane. Obie te bronie są tak zwaną bronią punktową. Karabiny mają zasięg punktowy, o którym będziemy dyskutować, ale w przypadku pistoletów i strzelb to naprawdę odnosi się do faktu, że w zakresach, w których te bronie są skuteczne, można rozsądnie oczekiwać trafienia tam, gdzie celujesz. Nie oznacza to, że bronie te nie muszą być celowane, aby były skuteczne, ale w szybkiej walce centralnej do większości strzelanek pierwszoosobowych byłoby nużące, gdyby gracz używał celowników na pistolecie, aby skutecznie uderzyć byle co. Zamiast tego, większość gier używa "strzelania z biodra" lub modelu swobodnego celowania, gdzie broń nie jest zgodna z aparatem. Ostrożny programista może jeszcze sprawdzić, czy cel znajduje się w skutecznym zasięgu pocisku, zanim zliczymy go jako trafienie, nawet jeśli automatycznie celujemy w ten sposób. Skuteczny zasięg to odległość, którą pocisk może pokonać przed uderzeniem w ziemię. Ustalenie tego jest prostym zastosowaniem równań w części 2 i części 6. Aby omówić broń palną, zaadaptowaliśmy kod z przykładu Cannon2 w Części 6, aby działał w programie Java o nazwie Marksman. W naszym przykładzie gracz patrzy przez lunetę na cel o długości 1 metra na 1 metr. Zapewniliśmy mu regulowany poziom powiększenia, więc wraz ze wzrostem zasięgu wciąż widzi cel. Punkt celowania jest przedstawiony jako puste kółko, a dziura po kuli jako okrąg wypełniony czarnym kolorem. Metoda, która została użyta do określenia, w którym kierunku celuje użytkownik w świecie modelu, przekształca lokalizację myszy w piksel na współrzędne w świecie modelu. Ta odległość i zakres są następnie wykorzystywane do znalezienia kątów wymaganych do celowania pistoletem. Kod do zrobienia jest pokazany dalej, gdzie alp i gmm są kątami nachylenia i łożyska. Są one mierzone w linii poziomej i w linii widzenia:

```
alp = 90-Math.toDegrees (Math.atan (((200-celny) * (celH / (drawH))) / zakres));

gmm = Math.toDegrees (Math.atan (((200-celX) * (targetH / (drawH))) / zakres));
```

gdzie targetH / drawH to tylko stosunek wysokości celu do wysokości celu w

piksele na ekranie. Pozwala to na przeliczenie współrzędnej myszy podanej w pikselach na metry. Następnie arcus tangens przelicza stosunek tych odległości na kąt dla pistoletu. Stała 200 odnosi się do układu współrzędnych pikseli, który znajduje się 200 pikseli od centrum celu. Jeśli chcesz dostosować to do pełnego systemu renderowania 3D, możesz usunąć wiele z tych konwersji, które musimy wykonać. Na 10-metrowym zasięgu pokazanym na rysunku 18-1 znajdujemy się w bliskiej odległości, a dziura po kuli pokrywa się z celownikami.

Na rycinie 18-2, z celem na 100 metrów, widzimy, że z jakiegoś powodu kula nie trafia w cel, w którym wycelowaliśmy broń. Na wysokości 300 metrów dziura po pocisku nie trafia nawet w cel. Widać, że po prostu rozważając ruch rzutowy i wyidealizowany opór, już mamy kłopot z trafieniem w dziesiątkę. Proces, w którym uwzględniane są te różnice, nazywa się zerowaniem celowników.

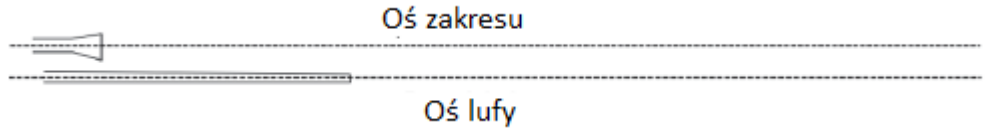
Zerowanie widoków

Idea zerowania celowników jest prawdopodobnie najważniejszą rzeczą do modelowania, jeśli chcesz mieć realistyczną strzelaninę w grze. Jak wspomnieliśmy wcześniej, kiedy gracze biegają od pokoju do pokoju, prawdopodobnie nie chcą myśleć o wietrze i zasięgu. Jednak w przypadku symulacji łowieckiej

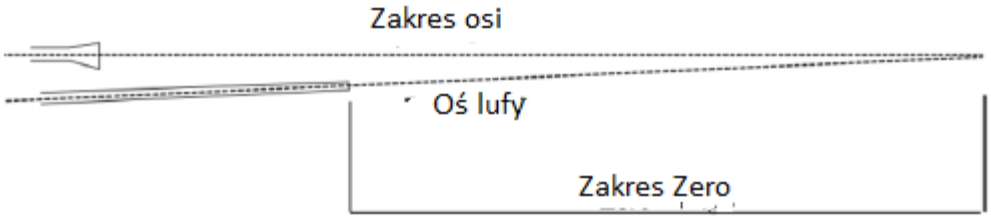
lub gry snajperskiej może być właściwe wprowadzenie tego. Kiedy dana osoba patrzy przez lunetę, jej ciało i karabin stają się sztywnymi ciałami, więc aby zmienić cel broni, musi ona obrócić całe swoje ciało. Jest to wygodne dla nas, ponieważ gracz zazwyczaj kontroluje pozycję strzelca lewą ręką za pomocą klawiatury i kierunku celowania prawą ręką. Inne metody celowania, krótko opisane wcześniej jako wolny cel, nie mają odpowiednika w świecie rzeczywistym, więc ograniczymy się do tego celu.

Kropla pocisku: Grawitacja i opór powietrza

Jeśli celujesz karabinem poziomo, możesz spodziewać się, że pocisk pozostawi wylot lufy w poziomie, a grawitacja i opór powietrza sprawią, że spadnie z niego. Rysunek 18-3 pokazuje kombinację karabinu i zakresu, która jest zamontowana idealnie równolegle. Ignorując wszystkie inne czynniki przez minutę, widzimy, że pocisk nigdy nie trafi tam, gdzie wskazany jest zasięg. Zawsze będzie kilka centymetrów nisko.



Dostosowując kontrolę elewacji zakresu, możemy sprawić, by karabin trafił tam, gdzie wskazany jest zasięg. (Patrz rysunek 18-4.)



Zakres, w którym pocisk przekroczy linię zdefiniowaną przez teleskop nazywany jest zakresem zerowym. Jeśli cel znajduje się w zakresie zerowym, wystarczy wskazać celownik i pociągnąć za spust. Jeśli usuniemy cel i wykreślimy trajektorię, będzie to wyglądało jak na rysunku 18-5.

Tutaj widzimy drugi punkt, w którym pocisk przekroczy linię utworzoną przez krzyżyk celownika. Nazywa się to dalekim zerem lub drugim zerem. To jest na ogół miejsce, w którym będzie strzelać. Ten schemat jest często dostępny u dostawców amunicji. Częściej oferują balistyczne stoły o niższych wysokościach. Ważne jest, aby korzystając z tych wykresów, które pamiętasz, zakładają zakres w poziomie i że punkt zaczyna się od ujemnej wysokości poniżej linii zasięgu. Tabela 18-1 pokazuje dane ze strony Remingtona i zakłada, że karabin został wyzerowany na 100 jardów; informuje, jak daleko pocisk znajduje się poniżej linii poziomej co 50 jardów po 0. Możesz ustawić współczynnik oporu w symulacji pocisku, aby dopasować te ustawieni

Zasięg na stoczniach	100	150	200	250	300	400	500
Spadek w centymetrach	0	-4.6	-13.8	-28,6	-50.1	-115,47	-219.1

Z tego stołu widać, że kropla pocisku ma ponad dwie stopy na 250 jardów. Oznacza to, że jeśli cel jest ustawiony na 250 jardów, a twój zasięg jest wyzerowany na 100 jardów, to musiałbyś celować dwa stopy powyżej celu, aby go trafić. To może być wystarczająco wysokie, aby nie można było już zobaczyć celu w tym zakresie! Aby przeciwdziałać temu problemowi, większość karabinów wyposażonych jest w lunety z korektami elewacji. To pokrętło z boku lunety, które można obracać do dyskretnych ustawień nazywanych kliknięciami. Większość zakresów używa 1/4 minuty regulacji kąta na kliknięcie, chociaż niektórzy używają 1/8, 1/2, a nawet pełnych minut. Minutę kąta to po prostu 1/60 stopnia. Dlatego, podczas dostosowywania do elewacji, strzelec może przekręcić gałkę na lunecie, a gdy usłyszy kliknięcia, wie, że dostosowała swój zasięg do wielu minut kąta. Teraz, gdy ponownie celuje w celownik celownika, lufa będzie nieco inna niż kąt, niż wcześniej, zasadniczo celując wyżej, aby pomieścić dłuższą odległość. Aby osiągnąć wysoką dokładność na polu, strzelec wiedział, że wyzerowała swój karabin do pewnego zakresu. Następnie, przy próbie strzału, oszacuje zasięg do celu i dostosuje zakres, jednak wiele kliknięć w górę lub w dół. Największą przyczyną błędu jest niedokładne oszacowanie zasięgu. Współcześni strzelcy często używają dalmierzy laserowych, aby dokładnie określić, jakie przesunięcie wysokości jest wymagane. W sytuacjach, w których fotografujesz na dużych odległościach, możesz zapewnić zasięg użytkownikom i pozwolić im dostosować zasięg karabinu z bieżącego zakresu zerowego do nowego zakresu zerowego. Większość dzisiejszych gier nie modeluje nawet wpływu grawitacji na pocisk, więc dodanie regulacji elewacji do snajpera lub części myśliwskiej w grze doda bardzo potrzebnej dokładności.

Wiatr

Podobnie jak w przykładzie pocisku z części 6, pociski naszej strzelonej cel dotknięty przez wiatr. Tak jak poprzednio, wrażliwość kuli na wiatr w dużym stopniu zależy od współczynnika oporu bocznego. W naszej symulacji można dostroić wrażliwość pocisku na wiatr, dostosowując współczynnik Cw. Dotyczy to również karabinów strzelających na długich dystansach. Na 20 metrach wiatr będzie miał niewiele do zrobienia w ustaleniu, gdzie trafi pocisk. Na wysokości 600 metrów może to spowodować, że pocisk będzie wyłączony o metr! Korekta dla wiatru jest podobna do regulacji wysokości. Przekręcając pokrętło, możesz dostosować zakres lub celownik tak, aby był nieco na prawo lub na lewo od linii środkowej beczki. Ten kąt, zwany gamma w naszej symulacji, pozwala zlikwidować efekt wiatru. Aby poradzić sobie z wiatrem w terenie, potrzebujemy prostych obliczeń, które zależą od konkretnego działania amunicji strzelca. Może działać w następujący sposób: jeśli wiatr wieje bezpośrednio w twoją kulę, ustawienie w calach będzie połowa prędkości wiatru w milach na godzinę. Oczywiście istnieje wiele innych zasad które różnią się dla każdego kalibru amunicji, ale dla naszej symulacji można dostroić współczynnik oporu wiatru do dowolnej wartości, jaką chce osiągnąć twój strzelec. Będzie musiał spędzić trochę czasu na dystansie, tak jak prawdziwy strzelec, zastanawiając się, jak bardzo wiatr wpływa na jego strzały. Gdy wiatr się zmieni, będzie musiał dostosować się w czasie rzeczywistym i albo celować w lewo, albo w prawo od celu lub dostosować ustawienia wiatru dla zakresu.

Oddychanie i pozycja ciała

Chociaż większość gier nie modeluje grawitacji i wiatru podczas obliczania trajektorii pocisku, wielu usiłuje uregulować dokładność, z jaką początkowo wystrzeliwujesz pocisk. Najczęściej twórcy gier osiągają to przez przybliżenie krzyżyków z czterema liniami, które się nie przecinają. Po wystrzeleniu pocisk trafi w dowolne miejsce wewnątrz koła opisanego przez wewnętrzne punkty końcowe tych czterech linii. Różne bronie mają różne dokładności, a linie mogą się poruszać, aby to odzwierciedlić. Zwykle pierwsze trafienie jest najdokładniejsze, a po wystrzeleniu broni musisz "odświeżyć" cel, a to wymaga czasu. Strzały wystrzeliwane w krótkim odstępie czasu stają się coraz mniej dokładne. W naszej symulacji modelowaliśmy kilka rzeczy, które wpływają na dokładność w rzeczywistym świecie i dajemy kilka sugestii dotyczących innych czynników, które można łatwo uwzględnić. Ponieważ nasza

gra najbardziej interesowała się strzelaniem dalekim za pomocą karabinów, najczęstszym źródłem błędów jest oddychanie. Jak wspomniano wcześniej, kiedy strzelec patrzy przez lunetę na lunetę, w zasadzie staje się stałym ciałem. Kiedy oddycha, karabin również oddycha. Podczas wykonywania trudnych ujęć, bardzo często zdarza się, że strzelec bierze oddech i przytrzymuje go podczas strzelania. W naszej symulacji modelowaliśmy to z klasą oddychania która dostosowuje punkt celu w górę i w dół wraz z upływem czasu, aby naśladować, jak porusza się zakres, gdy strzelający głęboko oddycha. Działa to za pośrednictwem licznika uruchomionego w naszym initComponents () uruchamia się co 100 ms. W poniższym kodzie zobaczysz to co prowadzi do oddechu co dwie sekundy.

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

timer.start ();
```

Funkcja ta powoduje przesunięcie punktu celu (celX, celY) niezależnie od kursor za pomocą następującego algorytmu:

```
if (direction == true) {

breathHeight = breathHeight

direction = false;

breathHeight = breathHeight + 1;

}

}

if (direction == false) {

breathHeight = breathHeight - 1;

if (breathHeight == -5) {

direction = true;

}

}

jeśli (oddychanie) {

aimY = aimY + breathHeight;

}
```

Tutaj po prostu przenosimy go do pewnego ograniczenia - w naszym przypadku 5 pikseli - a następnie przesuwając go z powrotem w dół. Lepsza implementacja zwiększyłaby niestabilność w miarę przybliżania się przez użytkownika, ponieważ wibracje są również powiększane. Nie ma ograniczeń co do innowacyjnych funkcji, za pomocą których można przesunąć kółko celowania z dala od kursora, aby zasymulować rzeczywistość celowania pistoletem. Jednak jest to z pewnością obszar strzelanek pierwszoosobowych, których brakuje. Kiedy jest gotowy do strzału, użytkownik może kliknąć lewym przyciskiem, aby wstrzymać oddech, zmienne oddechy staną się fałszywe, a celownik przestanie się poruszać. Ten prosty dodatek sprawia, że gra jest znacznie trudniejsza i bardziej wciągająca. Należy zauważyć, że jeśli strzelec wstrzyma swój oddech zbyt długo w prawdziwym życiu, cel znowu stanie się niestabilny, ponieważ jego ciało reaguje na brak świeżego tlenu. Kolejnym ulepszeniem tego algorytmu

byłoby, aby cel stał się niestabilny po naciśnięciu lewego przycisku myszy przez pewien czas. Wiele gier zmienia także dokładność broni w zależności od pozycji ciała. Istnieją trzy podstawowe typy pozycji strzeleckich: stojąca, klęcząca i skłonna. Stanie to - odgadłeś - stojąc. Klęczenie to raczej forma przysiadu niż klęczenie na kolanach. Prone leży płasko na ziemi. Ponieważ karabin jest zamknięty na ciele, im mniej niestabilne, tym mniej niestabilny cel. Kiedy stoisz, mięśnie twojego ciała muszą dużo pracować, aby pozostać w pozycji pionowej. Kiedy klęczą, robią mniej, a kiedy są podatne, twoje mięśnie nie muszą martwić się o to, że będziesz stać w ogóle. Możesz dodać te parametry jako losowe drgania w celu i dostroić je, aby zmienić względną przewagę każdej pozycji. Jednak skłonność powinna zawsze być bardziej stabilna niż klękanie, a klękanie bardziej stabilne niż stanie.

Odrzut i wpływ

Teraz, gdy celowaliśmy, strzelaliśmy i zorientowaliśmy się, gdzie jest kulka w danym momencie, porozmawiajmy o ostatniej fazie, końcowej balistyce. Aby naprawdę zrozumieć, co dzieje się na końcu lotu kuli, powróćmy do początku. Wcześniej rozmawialiśmy o odrzutu w wyniku zachowania dynamiki Newtona. Wszyscy widzieli kiepski film, w którym bohater strzela do złego faceta, a kule powodują, że zły facet zostaje zdmuchnięty z nóg. Z tym jest duży problem! Gdyby pociski były wystarczająco potężne, by powalić osobę, z którą uderzyły jej stopy, to osoba strzelająca z pistoletu również zostałaby zdmuchnięta z nóg! W rzeczywistości siła odczuwalna przez strzelającą osobę jest prawie taka sama jak siła odczuwalna, gdy broń się cofa. W przypadku pocisku 9 mm ważącego 7,45 g, który opuszcza lufę z prędkością 390 m / s, broń odczuje odrzut tak, że jej moment jest równy momentowi pocisku. Ciekawym sposobem na włączenie odrzutu do gry wideo jest kosmos. Na Ziemi odrzut broni jest dość szybko przenoszony na ziemię poprzez tarcie między graczem a wielką złą ziemią. W kosmosie strzelec nie ma korpusu planetarnego, który mógłby naciskać, więc odrzut pistoletu staje się odrzutem układu pistolet / osoba. Następnym razem, gdy twoja postać będzie musiała przenieść się z jednego statku do drugiego w środowisku mikro-grawitacyjnym, możesz zmusić ją do wydania amunicji, aby mogła się poruszać. Teraz, jeśli zostaniesz postrzelony, prawdopodobnie szybko upadniesz, ale ma to więcej wspólnego z biologią niż z fizyką. Jednak ignorując żywe cele, jeśli chcesz symulować obrażenia zadane przez trafienie w coś, ważniejsze jest spojrzenie na energię kinetyczną pocisku. W rzeczywistości pociski i pociski artyleryjskie nazywane są bronią kinetyczną, ponieważ ich głównym środkiem niszczenia celu jest przeniesienie ich energii kinetycznej na cel. Jest inaczej niż, powiedzmy, bomba, która przenosi swoją energię chemiczną na ciepło i energię kinetyczną po uderzeniu.

Eksplozje

Dokładne modelowanie eksplozji obejmuje wielofizyczne symulacje płynów, takie jak te omówione w części 14 do rozdziału 16. Jedną z naszych gier domowych jest to, że gry wideo zwykle mają kolekcję beczek leżących wokół tego, jeśli zrobione raz, eksplodują wystarczająco gwałtownie, by wysadzić pobliskie pojazdy. Chociaż pozwala to na łatwą walkę z wieloma wrogami, tak naprawdę trudno jest zdmuchnąć przedmioty codziennego użytku. Strzelanie z pistoletu gazowego prawie nigdy nie spowoduje pożaru, a tym bardziej wybuchu. Rzeczywiście, nawet strzelanie do zbiornika z propanem z karabinem nie da ci fajerwerków. Wystarczyłoby coś w rodzaju zbiornika z 1/4 propanu zmieszanego z tlenem 3/4, aby wybuchł, a te zwykle nie leżą w pobliżu. Niezależnie od tego, kiedy gramy w gry wideo, często jesteśmy wdzięczni, że mamy sporą czerwoną lufę do robienia zdjęć, więc sprawdzimy, jak sprawić, by powstały wybuch był dokładniejszy, nawet jeśli zapłon jest nieprawdopodobny.

Eksplozje cząstek

W przypadku większości eksplozji w grze wystarczy zaimplementować eksplozję typu cząstek, którą omówiliśmy w rozdziale 2. Teraz, w rozdziale 2, cząstki były po prostu kropkami, ale nie muszą być ograniczone do tak prostych spritów W niektórych przypadkach, takich jak iskrzenie z pocisku

uderzającego w metalowy pojemnik, bardzo dokładne byłoby modelowanie eksplozji jako cząstek; Jednak dzięki temu, że nasze cząstki wyglądają jak fragmenty samochodów, możemy również sprawić wrażenie, że sam samochód eksplodował. Jest to łatwiejsze, ponieważ eksplozje cząstek nie mają żadnego ruchu kątowego. Chociaż możesz przypisać różne części samochodów różnym cząstkom, kiedy odlecą z powodu eksplozji, nie będą się obracać. Dobra wiadomość jest taka, że eksplozja cząstek nadal da ci realistyczną dystrybucję fragmentów czegoś na ziemi. Aby porozmawiać o tym, jak połączyć pociski i eksplozje cząstek, omówimy coś bardziej fizycznie dokładna niż pocisk wysadzający w powietrze samochód. Rozważmy kulę uderzającą w jakiś luźny żwir. Zwykle spowoduje to wyrzucenie żwiru w powietrze z kolizji pocisku. Zamiast próbować obliczać skomplikowane kolizje podczas uderzenia, wygenerujemy wybuch cząstek w oparciu o kod w części 2:

```
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;

}

}
```

Jak widać, początkowa prędkość V0 kontroluje siłę wybuchu. W części 2 wybraliśmy tę wartość losowo. Teraz, kiedy mamy kulę lecącą w powietrzu, możemy lepiej oszacować siłę wybuchu. Jak sobie przypominasz z wcześniejszej części, pocisk ma energię związaną z nim w dowolnym czasie, t, w locie. Ta energia jest jej energią kinetyczną i jest równa połowie masy pocisku razy jej prędkość do kwadratu. W naszej symulacji pocisku łatwo jest obliczyć tę energię, gdy kula leci w powietrzu. Należy zauważyć, że wielka kula poruszająca się powoli jest tak samo potężna jak mniejszy pocisk poruszający się szybko. Nasz nadchodzący kod zakłada, że 100% energii kinetycznej jest dostarczane do celu. Nie byłoby to prawdą, gdyby kula trafiła prosto w coś. Można sobie wyobrazić dwa cele, oba wiszące na suficie. Jeden wykonany jest z papieru, a drugi ze stali. Po trafieniu ,stalowy cel kołysze się z jego podpory, a papierowy cel pozostaje nieruchomy. Dzieje się tak dlatego, że kula przechodzi prosto przez papier i nie przenosi swojej energii kinetycznej na cel. Aby wszystko było proste, przeniesiemy całą energię kinetyczną pocisku na żwir. W formie równania wyglądałoby to tak:

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

Zauważ, że jest to suma poszczególnych bitów prędkości żwiru. W części 2 każda cząsteczka otrzymała losową prędkość w dowolnym miejscu od Vinit / 2 do Vinita. Może to doprowadzić do stworzenia zestawu cząstek, których energie przekraczają energię wejściową. Aby temu zapobiec, dodamy zmienną do naszej klasy wybuchu, jak na przykład:

```
typedef struct _TParticle

{

float x; // x-coordinate of the particle

float y; // y-coordinate of the particle

float vi; // initial velocity

float angle; // initial trajectory (direction)

int life; // duration in milliseconds

int r; // red component of particle's color
```

```
int g; // green component of particle's color
int b; // blue component of particle's color
int time; // keeps track of the effect's time
float gravity; // gravity factor
BOOL Active; // indicates whether this particle
// is active or dead
float mass; //for calculating the particle's energy
} TParticle;
#define _MAXPARTICLES 50
#define _MASSOFPARTICLE .25
typedef struct _TParticleExplosion
{
TParticle p[_MAXPARTICLES]; // list of particles
// making up this effect
int x; // initial x location
int y; // initial y location
float KE; //Available kinect energy
float
BOOL Active; // indicates whether this effect is
//active or dead
} TParticleExplosion;
```

Zauważ, że V0 nie jest już potrzebny, ponieważ dostępna energia kinetyczna będzie decydować o sile eksplozji. Zakładając, że energia kinetyczna pocisku zostanie podana jako zmienna KEb, nasza nowa funkcja CreateParticleExplosion będzie wyglądała następująco:

```
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; //Mass of a single gravel
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;
}
}
```

Jak widać, zmieniliśmy zdania, które określają początkową prędkość cząstek jako generatora liczb losowych w zakresie od 0 do prędkości, która pochłania energię kinetyczną całego wybuchu. Następna linia zmniejsza dostępną energię kinetyczną w eksplozji o wartość właśnie przypisaną cząstce. W ten sposób możesz być pewny, że wychodząca eksplozja nigdy nie jest mocniejsza niż dane wejściowe. Bardziej interesującym sposobem radzenia sobie z tym byłoby inicjowanie cząstek z pewnym określonym rozkładem masy i przypisywanie prędkości nie losowo, ale z rozkładem normalnym. Numeryczne przepisy w języku C mogą pomóc w osiągnięciu tego. Mimo że powyższy kod nie uwzględnia niektórych z bardziej subtelnych aspektów transferu energii kinetycznej, zapewni to, że mały, wolno poruszający się pocisk wytwarza mniejszą eksplozję niż duża, szybko poruszająca się. To jest coś, czego brak w dzisiejszych gier wideo.

Eksplozje Wielokątów

Podczas eksplozji cząstek są odpowiednie dla małych, jednolitych obiektów, ale nie dają odpowiedniego realizmu, gdy coś jest wdmuchiwane w identyfikowalne kawałki. Dlatego w grach wideo rzadko widzisz, jak samochód eksploduje, a drzwi odlatują, by wylądować obok ciebie. Zamiast tego, gry zazwyczaj obsługują takie obiekty z eksplozją cząstek, która zasłania obiekt, gdy jest ponownie renderowany w stanie rozłożonym na nowo, a brakujące części zostały najwyraźniej rozerwane na strzępy. Jeśli chcesz modelować pełną eksplozję ciał stałych, możesz ponownie użyć kodu cząstkowego do aspektów tłumaczenia. Zasadniczo cząstki będą teraz opisywać środek ciężkości każdego ciała stałego. Będziesz musiał dodać początkową prędkość kątową i pozwolić, aby symulacja, zgodnie z opisem w części 12, obsługiwała ich ruch po tym początkowym kącie. Chociaż nie mamy miejsca, aby przejść do innego przykładu tutaj, porozmawiamy trochę o energii wejściowej do takiej eksplozji, aby pomóc w wypełnieniu luki. Kiedy zajmujemy się tym tematem, przypomnijmy sobie, że kula nie ma wystarczającej energii, aby rozniesć coś na strzępy. Nawet jeśli uderzysz czymś w działo czołgowe, to naprawdę nie jest kinetyka W przypadku uderzenia czołgu w inny zbiornik, stopiony żużel z uderzenia jest zwykle zasypywany wewnątrz zbiornika, powodując paliwo lub amunicja do wybuchu. To tam dostajecie wielkie bumy - to konwersja energii chemicznej na ciepło, światło i ciśnienie! Najpopularniejsza metoda określania ilości energii chemicznej w broni jest nazywana Równoważnością TNT. To tyle TNT potrzebował by wywołać tę samą eksplozję niezależnie od tego, co faktycznie eksplodujesz. Teraz modelowanie wybuchu, powiedzmy, benzyny i powietrza jest dość skomplikowane, więc trzymajmy się TNT. Kilogram TNT zawiera 4.184 Mega dżuli energii; runda 9 mm ma 400 J. Z tego porównania widać, dlaczego ciężko jest wysadzić w powietrze, strzelając do niego, ale łatwo zrobić z blokiem TNT. Na potrzeby tej dyskusji powiedzmy, że masz otwarte pole (pięć boków wielokąta), na które gracz rzucił tylko 1 kg bloku TNT. Kiedy TNT jest detonowane, możesz nadać każdej stronie wieloboku prędkość początkową (translacyjną i kątową) i pozwolić na przejęcie równań kinematycznych. Te prędkości mogą opierać się na dwóch prostych zasadach.

• Wektor prędkości można zdefiniować za pomocą dwóch punktów: środka bloku TNT i środka obszaru wielokąta.

• Suma całej energii kinetycznej musi być mniejsza niż dostępna energia chemiczna w TNT. Może to być proporcjonalne do kwadratu odległości od wielokąta do bloku TNT.

Użycie środka obszaru w naszej pierwszej zasadzie spowoduje pewien obrót w naszym wielokącie, ponieważ spowoduje to moment wokół środka ciężkości, chyba że oba się pokryją. W takim przypadku, tak jak w przypadku naszego pudła, opór aerodynamiczny i nierówności eksplozji będą nadal powodować obrót, więc należy albo modelować je jawnie, albo przekazać ręcznie prędkość obrotową. Teraz, gdy mamy kierunek prędkości, musimy zdefiniować jego wielkość. Siła działająca na obiekty znajdujące się w pobliżu wybuchu jest spowodowana szybkim rozszerzaniem się gazów w wyniku ciepła

generowanego przez detonację materiału wybuchowego. Jednak nie cała energia chemiczna jest przenoszona na obiekty - wiele z nich zamienia się w ciepło, światło i dźwięk. Zazwyczaj tylko jedna trzecia dostępnej energii chemicznej jest przekształcana w początkowej detonacji. Nazwijmy to wydajnością wybuchu, którą oznaczymy przez ζ. Dlatego możemy zapisać zależność między prędkościami wielokątów w następujący sposób:

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

Tutaj możemy dostroić ζ, aby nadać wielokątom realistyczne prędkości w przypadku wybuchu (tj. Nie wysyłając ich z prędkością światła). Jeśli energia z wybuchu jest równo podzielona, można zobaczyć, że lżejsze obiekty będą miały wyższe prędkości, jak można się spodziewać. Możesz także dostosować ilość energii eksplozji dostępnej dla każdego obiektu, ważąc energię przekazywaną przez obiekt w odległości od eksplozji. Powinno to być również dostrojone w programie, ale ogólnie ciśnienie eksplozji zmniejsza się wraz z sześcianem odległości i wykładniczo wraz z upływem czasu. Jeśli chcesz modelować bardziej złożone interakcje między eksplozjami i strukturą, istnieje wiele dobrych referencji dotyczących tego, jak, powiedzmy, budynków, reagować na wybuchy bomb. FEMA, a także armia i marynarka wojenna, mają kilka artykułów na ten temat, takich jak podręcznik referencyjny FEMA 426, aby zminimalizować potencjalne ataki terrorystyczne przeciwko budynkom. Ogólne pojęcia zawarte w takich dokumentach mogą zwiększyć realizm szkód budowlanych spowodowanych eksplozjami.