XIV. Silniki Fizyki

Silnik fizyki jest częścią twojej gry, która zawiera cały kod wymagany do tego, co próbujesz symulować za pomocą technik opartych na fizyce. Dla wielu programistów gier silnik fizyki jest symulatorem sztywnego ciała w czasie rzeczywistym, takim jak ten, o którym mówiliśmy wcześniej w tym tekście. Dostępne i dostępne do licencjonowania silniki fizyki dostępne są zazwyczaj w wersji o sztywnym nadwoziu. Niektóre silniki fizyki są raczej ogólne i są użyteczne dla ogólnych sztywnych ciał i cząstek; inne zawierają różne złącza i ograniczenia, umożliwiające symulację ragdoll. Jeszcze inne koncentrują się na miękkich ciałach i płynach. Mniej koncentruje się na fizyce konkretnej rzeczy, takiej jak samochód czy łódź. Proste wyszukiwanie w Internecie na frazie "silnik fizyki gry" wygeneruje wiele linków do potencjalnych opcji do użytku. Powiedziałeś, że zawsze możesz napisać własny silnik fizyki.

Budowanie własnego silnika fizyki

Jesteśmy zwolennikami korzystania z fizyki tam, gdzie jej potrzebujesz. Oczywiście, możesz napisać ogólny, fizykalny silnik gry, ale jeśli tworzysz grę, która nie wymaga silnika fizyki ogólnego przeznaczenia, nie pisz tego. To może wydawać się oczywiste, ale czasami jesteśmy zmuszeni robić więcej, niż potrzebujemy, abyśmy mogli powiedzieć, że to zrobiliśmy. Poza wysiłkiem fizyczny silnik fizyki ogólnego przeznaczenia będzie prawdopodobnie mniej wydajny niż skonstruowany specjalnie do tego celu silnik fizyki. Zbudowany celowo, mamy na myśli projektowanie silnika fizyki specjalnie do tego, co próbujesz symulować. Na przykład silnik fizyki ogólnego przeznaczenia z pewnością zawierałby cząstki, ciała sztywne, złącza, inne efektory siły i kto wie, co jeszcze - może płyny - i może być w pełni 3D. Ale jeśli piszesz grę 2D przewijaną w stronę smartfona, na pewno nie będziesz potrzebował 3D z powiązanymi złożonościami związanymi z obrotami i kolizjami w 3D; a jeśli twoja gra polega po prostu na rzucaniu kulą w jakieś arbitralne rupiecie, to w ogóle nie musisz nawet radzić sobie ze sztywnymi ciałami. Trochę tu kpiąco, ale chodzi o to, że chyba musisz napisać silnik fizyki ogólnego przeznaczenia - powiedzmy, jeśli masz zamiar licencjonować go jako produkt warstwy pośredniej lub używać go w różnych typach gier - wtedy nie rób tego. napisz jeden. Zamiast tego napisz specjalnie zoptymalizowany pod kątem gry, nad którą pracujesz. Rozważmy kilka przykładów. Załóżmy, że piszesz strzelankę 3D z perspektywy pierwszej osoby i chcesz użyć fizyki do symulowania, jak drewniane lufy i skrzynie rozrywają się podczas strzału. Zazwyczaj taki efekt pokazywałby kawałki drewna wylatujące w różnych kierunkach pod wpływem grawitacji. Możesz symulować taki efekt w 3D za pomocą sztywnych ciał, a nie musisz nawet brać pod uwagę kolizji, chyba że chcesz, aby kawałki odbijały się od siebie nawzajem lub innych obiektów. Ignorowanie tych aspektów znacznie upraszcza podstawowy silnik fizyki. Rozważ kolejny przykład. Załóżmy, że pracujesz nad grą polegającą na lataniu samolotem. Możesz użyć fizyki do symulacji dynamiki lotu, jak wyjaśnimy w tej książce, bez potrzeby użycia cząsteczek, złączy, a nawet reakcji kolizji. Celem całej tej dyskusji jest to, że powinieneś rozważyć, które aspekty twojej gry naprawdę skorzystają na fizyce i napisać twój silnik fizyki, aby poradzić sobie z tymi aspektami. Kolejną kwestią do rozważenia jest to, czy potrzebujesz fizyki w czasie rzeczywistym. Można się spodziewać, po przeczytaniu dostępnej literatury fizyki gry, że gra musi zawierać symulacje w czasie rzeczywistym, jeśli ma ona uwzględniać fizykę. Jest jednak wiele sposobów na włączenie fizyki do gry bez konieczności rozwiązywania fizyki za pomocą symulacji w czasie rzeczywistym. Pokażemy przykład w Części 19, gdzie symulowana jest huśtawka golfowa w celu ustalenia prędkości główki maczugi w momencie uderzenia piłki klubowej. W tym przypadku, biorąc pod uwagę określone parametry początkowe, możemy szybko i praktycznie błyskawicznie rozwiązać problem, aby określić prędkość maczugi, która może być następnie wykorzystana jako początkowy warunek lotu piłki. Lot piłki może być szybko rozwiązany, niekoniecznie w czasie rzeczywistym. To naprawdę zależy od tego, jak chcesz zaprezentować wynik graczowi. Jeśli twoja gra polega na podążaniu za lotem piłki, która unosi się w powietrzu, możesz symulować jej lot w czasie rzeczywistym, aby realistycznie przesuwać piłkę i kamerę. Jeśli jednak chcesz

po prostu pokazać, gdzie kończy się kula, nie musisz wykonywać symulacji w czasie rzeczywistym. W przypadku tak prostego problemu, możesz rozwiązać ostateczną lokalizację piłki szybciej niż w czasie rzeczywistym. Czasami akcja, którą symulujesz, może nastąpić tak szybko w prawdziwym życiu, że chcesz ją spowolnić w grze. Po locie piłeczki golfowej w czasie rzeczywistym kamera może poruszać się tak szybko, że gracz nie będzie mógł cieszyć się pięknym widokiem z lotu ptaka. W takim przypadku szybko rozwiązujesz trasę lotu, zapisujesz dane, a następnie animujesz scenę w przyjemniejszym tempie, jaki wybierzesz. Nie chcemy natknąć się na próbę wypominania cię o pisaniu silnika fizyki, jeśli tak zdecydujesz. Punktem naszej dotychczasowej dyskusji jest to, że po prostu nie musisz pisać generycznego silnika fizyki działającego w czasie rzeczywistym, aby używać fizyki w swoich grach. Masz inne opcje, jak właśnie wyjaśniliśmy. Zakładając, że po tych wszystkich rozważaniach, trzeba napisać silnik fizyki, a następnie mamy do zaoferowania następujące. Silnik fizyki jest tylko jednym z elementów silnika gry. Pozostałe komponenty obejmują silnik graficzny, silnik audio, silnik sztucznej inteligencji i wszelkie inne wymagane silniki lub inne elementy gry, które możesz podnieść do stanu silnika. W każdym razie silnik fizyki radzi sobie z fizyką. W zależności od tego, z kim rozmawiasz, dostaniesz różne pomysły na komponowanie silnika fizyki. Niektórzy powiedzą, że sercem silnika fizyki jest moduł wykrywania kolizji. A co, jeśli twoja gra nie wymaga wykrywania kolizji, ale nadal wykorzystuje fizykę do symulacji określonych zachowań lub funkcji? Wtedy wykrywanie kolizji z pewnością nie może być sercem twojego silnika fizyki. Niektórzy programiści z pewnością będą mieli zastrzeżenia do tych stwierdzeń. Dla nich silnik fizyki symuluje ruch sztywnego ciała za pomocą dynamiki Newtona, jednocześnie dbając o wykrywanie kolizji i reakcję. Dla nas istotnym składnikiem silnika fizyki jest model - to znaczy idealizacja rzeczy, którą próbujesz symulować w realistyczny sposób. Nie można realistycznie symulować charakterystyki lotu określonego samolotu, traktując go jako ogólny sztywny korpus. Musisz opracować reprezentatywny model tego samolotu, w tym bardzo specyficzne cechy; w przeciwnym razie jest to hack (który, nawiasem mówiąc, uznajemy za poprawne i długotrwałe podejście). Wcześniej, w części 7 i części 13, pokazaliśmy kilka przykładowych symulacji. Te proste przykłady zawierają wiele wymaganych komponentów ogólnego silnika fizyki. Istnieją klasy cząstek i ciała sztywnego, które obejmują ogólne właściwości i zachowania obiektów, modele fizyczne, które regulują zachowania obiektów, systemy wykrywania kolizji i odpowiedzi oraz numeryczny integrator. Ponadto przykłady te obejmują łączenie kodu fizycznego z danymi wprowadzanymi przez użytkownika i wizualnymi informacjami zwrotnymi. W tych przykładach pokazano również podstawowy przepływ z danych wprowadzanych przez użytkownika do rozwiązania fizyki do wizualnego sprzężenia zwrotnego. Podsumowując, główne elementy generycznego silnika fizyki to:

• Modele fizyki

• Menedżer symulowanych obiektów

• Silnik wykrywania kolizji lub jego interfejs

• Moduł odpowiedzi na kolizję

• Efekty siły

• Integrator numeryczny

• Interfejs silnika gry

Modele fizyki

Modele fizyki to idealizacje rzeczy, które symulujesz. Jeśli twoja fizyka Silnik to generyczny symulator sztywnego ciała, wykorzystywany do symulacji asortymentu obiektów pełnych, które gracze mogą pokonywać, rzucać, strzelać i na ogół wchodzić w interakcje z nimi w podstawowy sposób, wtedy model

fizyczny będzie prawdopodobnie bardzo ogólny. Prawdopodobnie można bezpiecznie założyć, że każdy obiekt będzie podlegał przyciąganiu grawitacji, a więc masa będzie ważnym atrybutem. Rozmiar będzie również ważny, nie tylko dlatego, że będziesz musiał wiedzieć, jak duże rzeczy są podczas sprawdzania kolizji i obsługi innych interakcji, ale także dlatego, że rozmiar jest związany z rozkładem masy obiektu. Dokładniej, każdy obiekt będzie miał atrybuty bezwładności masy. Obiekty najprawdopodobniej będą miały również pewien przypisany współczynnik restytucji, który będzie używany podczas obsługi kolizji. Dodatkowo możesz przypisać niektóre współczynniki tarcia, które mogą być używane podczas reakcji kolizji lub w sytuacjach, gdy obiekty mogą ślizgać się po podłodze. Ponieważ obiekty prawdopodobnie znajdą się w powietrzu w pewnym momencie, prawdopodobnie uwzględnisz również współczynnik oporu dla każdego obiektu. Wszystkie te parametry pomogą odróżnić obiekty masywne od lżejszych lub obiekty kompaktowe od dużych. Jeśli twoja symulacja obejmuje więcej niż ogólne sztywne ciała, twoja fizyka będzie bardziej konkretna i może znacznie bardziej rozbudowana. Doskonałym przykładem bardziej skomplikowanego modelu jest symulacja lotu. Bez względu na to, jak dobry jest Twój ogólny sztywny model nadwozia, nie będzie latać jak żaden konkretny samolot, jeśli leci w ogóle. Musisz opracować model, który przechwytuje aerodynamikę lotu charakterystyczną dla symulowanego samolotu. Część 15 pokazuje, jak stworzyć model samolotu, który może być użyty w symulacji lotu w czasie rzeczywistym. Pozostałe rozdziały mają dać ci smak aspektów modelowania różnych rzeczy, które możesz symulować w grze. Podobnie jak nie można symulować samolotu z typowym modelem o sztywnym nadwoziu, nie można symulować statku z modelem samolotu, ani nie można symulować piłki golfowej za pomocą modelu statku. Chodzi o to, że musisz poświęcić trochę czasu na zaprojektowanie swojego modelu fizycznego, specyficznego dla tego, co będziesz symulował w grze. Czas spędzony tutaj jest równie ważny dla stworzenia realistycznego silnika fizyki, jak czas spędzony na zaprojektowaniu solidnego schematu integracji lub systemu wykrywania kolizji. Nie możemy przecenić znaczenia modelu fizycznego. Model jest tym, co definiuje zachowanie rzeczy, którą symulujesz.

Menedżer Symulowanych Obiektów

Menedżer symulowanych obiektów będzie odpowiedzialny za tworzenie instancji, inicjowanie i usuwanie obiektów. Będzie również odpowiedzialny za utrzymywanie powiązań między fizyką obiektu a innymi atrybutami, takimi jak geometria, na przykład, jeśli w symulacji 3D używasz tego samego wielościanu do renderowania obiektu i wykrywania kolizji i odpowiedzi. Musisz mieć możliwość zarządzania obiektami w twojej symulacji. Można sobie wyobrazić wiele różnych podejść do zarządzania tymi obiektami, i chyba że twoja symulacja wykorzystuje tylko garść obiektów lub mniej, w zasadzie potrzebna jest lista obiektów o dowolnej klasie, którą zdefiniowałeś. Widzieliście w przykładach z poprzednich rozdziałów, gdzie używamy prostych tablic obiektów typu RigidBody lub obiektów typu Particle. Jeśli wszystkie obiekty w twojej symulacji są takie same, potrzebujesz tylko jednej klasy przechwytującej wszystkie ich zachowania. Jednak dla większej różnorodności powinieneś użyć listy różnych klas z każdą klasą zawierającą kod wymagany do wdrożenia własnego modelu fizycznego. Jest to szczególnie ważne w odniesieniu do sił działających na model. Na przykład możesz mieć obiekty reprezentujące pociski z innymi samolotami. Te różne klasy będą miały wspólny kod (na przykład wykrywanie kolizji); jednak sposób obliczania sił na każdym będzie się różnić ze względu na różnice w sposobie ich modelowania. Przy takim podejściu każda klasa musi mieć kod, który implementuje swój konkretny model. Podczas integracji cała lista zostanie przetransponowana, wywołując metodę agregacji sił dla każdego obiektu, a dana klasa zajmie się szczegółami odpowiednimi dla danego typu obiektu. W niektórych prostych przypadkach nie trzeba używać innych klas obiektów, jeśli typy obiektów, których planujesz użyć, nie różnią się zbytnio. Na przykład byłoby całkiem proste zaimplementowanie jednej klasy zdolnej do obsługi zarówno cząstek, jak i ciał sztywnych. Klasa obiektu może zawierać właściwość typu obiektu używana do oznaczenia, czy obiekt jest cząsteczką lub ciałem

sztywnym, a następnie metody klasy wywoływałyby odpowiedni kod. Ponownie, to będzie działać poprawnie dla prostych obiektów z niewielkimi różnicami. Jeśli chcesz zasymulować więcej niż dwa typy obiektów lub są one bardzo różne, prawdopodobnie lepiej jest zastosować różne klasy specyficzne dla każdego symulowanego obiektu. Jakkolwiek uporządkujesz swoje klasy lub listy, przepływ przetwarzania twoich obiektów będzie na ogół taki sam. Każdy tyk fizyki - czyli za każdym razem w symulacji fizyki - musisz sprawdzić kolizje obiektów, rozwiązać te kolizje, zsumować zwykłe siły na każdym obiekcie, zintegrować równania ruchu dla każdego obiektu, a następnie zaktualizować stan każdego obiektu. Jak powiedzieliśmy, jest to ogólny przepływ na każdym teście fizyki lub kroku czasu, który może nie być taki sam jak kroki renderowania. Na przykład dla dokładności w twojej symulacji możesz potrzebować małych kroków około milisekundy. Nie chcesz aktualizować grafiki co milisekundę, gdy potrzebujesz tylko około jednej trzeciej aktualizacji grafiki na sekundę. W ten sposób menedżer obiektów będzie musiał być zintegrowany z ogólnym mechanizmem gry, a silnik gry musi być odpowiedzialny za upewnienie się, że fizyka i grafika są odpowiednio aktualizowane.

Wykrywanie kolizji

Jeśli kolizje są ważną częścią twojej gry, wymagany jest solidny system wykrywania kolizji. Twój system wykrywania kolizji różni się od systemu reagowania na kolizje lub modułu, chociaż te dwa elementy idą w parze. Wykrywanie kolizji jest problemem geometrii obliczeniowej polegającym na tym, że zderzają się obiekty, a jeśli tak, z jakimi punktami stykają się. Punkty te są czasami nazywane kolektorem kontaktowym. Są to tylko punkty, które się dotykają, które mogą być liniami lub powierzchniami, ale dla uproszczenia zwykle punkt, punkty końcowe lub punkty definiujące granicę powierzchni styku są zawarte w kolektorze stykowym. Rola systemu wykrywania kolizji jest bardzo specyficzny: określa, które obiekty kolidują, jakie punkty na każdym obiekcie są zaangażowane w kolizję i prędkości tych punktów. Brzmi prosto, ale rzeczywista implementacja może być dość skomplikowana. Są sytuacje, w których szybko poruszające się obiekty mogą przechodzić w prawo przez inne obiekty, szczególnie cienkie, w jednym kroku czasowym, co powoduje, że system wykrywania kolizji omija kolizję, jeśli polega wyłącznie na sprawdzeniu odległości separacji między obiektami i ich względnej prędkości (tj. , wykrywa kolizję, jeśli obiekty znajdują się w granicach tolerancji kolizji i poruszają się także względem siebie). Solidny system wykrywania kolizji pozwoli uchwycić tę sytuację i odpowiednio zareagować. W części 8 po prostu sprawdzamy, czy cząstki przesuwają się obok ziemi w trakcie pojedynczego kroku czasowego, a następnie przestawiamy ich położenie na poziom płaszczyzny podłoża. Potrafimy radzić sobie w wielu sytuacjach przy użyciu takich prostych technik, szczególnie w przypadku obiektów przechodzących przez podłogi lub ściany; jednak inne sytuacje mogą wymagać bardziej złożonych algorytmów do przewidywania, czy kolizja wystąpi kiedyś w niedalekiej przyszłości, w zależności od tego, jak szybko obiekty poruszają się względem siebie. Ten ostatni przypadek nazywa się ciągłym wykrywaniem kolizji i znajduje się w wielu źródłach internetowych, książkowych i technicznych. Wiele komercyjnych i fizycznych silników typu open source reklamuje swoją zdolność do obsługi ciągłego wykrywania kolizji. Kolejnym wyzwaniem związanym z wykrywaniem kolizji jest fakt, że może to być bardzo czasochłonne, jeśli masz dużą lub nawet umiarkowaną liczbę obiektów w swojej symulacji. Istnieją różne techniki radzenia sobie z tym. Po pierwsze, przestrzeń do gry jest podzielona na pewne grube sieci, a ta siatka służy do porządkowania obiektów w zależności od tego, którą komórkę zajmują. Następnie w drugiej fazie wykrywania kolizji sprawdzane są tylko te obiekty, które zajmują sąsiednie komórki, aby sprawdzić, czy zderzają się. Bez tego podziału na partycje, sprawdzanie parami każdego obiektu względem każdego innego obiektu byłoby bardzo kosztowne obliczeniowo. Druga faza wykrywania kolizji jest często szerokim podejściem przy użyciu sfer granicznych lub obwiedni, które mogą być wyrównane względem osi lub ciała. Jeśli graniczne kule lub pudła każdego obiektu zostaną zderzone, to obiekty te mogą również kolidować i będą wymagane dalsze kontrole; w przeciwnym razie możemy wywnioskować, że obiekty nie kolidują

ze sobą. W przypadku potencjalnej kolizji, te dalsze kontrole stają się bardziej złożone w zależności od geometrii obiektów. Ta faza zazwyczaj obejmuje kontrole na poziomie wielokąta i wierzchołka; istnieją sprawdzone techniki przeprowadzania takich kontroli, których nie będziemy tutaj wprowadzać. Ponownie, w Internecie dostępna jest bogata literatura dotycząca wykrywania kolizji.

Odpowiedź kolizji

Gdy system wykrywania kolizji wykona swoje zadanie, nadszedł czas, aby system reagowania na kolizje poradził sobie z kolizją obiektów. Wcześniej w tej książce pokazaliśmy, jak wdrożyć metodę kolizji impuls-moment. Przypomnijmy, że ta metoda zakłada, że w momencie kolizji, najbardziej znaczące siły działające na obiekty są siłami kolizyjnymi, więc wszystkie inne siły mogą zostać zignorowane w tym momencie. Metoda następnie oblicza uzyskane prędkości obiektów po zderzeniu i natychmiastowo zmienia ich prędkości. Aby wykonać wymagane obliczenia, system reagowania na kolizje wymaga kolizji obiektów, oczywiście, punktów kolizji i prędkości tych punktów. Ponadto każdy obiekt musi mieć pewną powiązaną masę i współczynnik restytucji, które są również wykorzystywane do obliczania wynikowych prędkości obiektów po kolizji. W praktyce system reagowania na kolizje działa ręka w rękę z systemem wykrywania kolizji, szczególnie w przypadku obiektów, które mogą się przeniknąć. Jak wspomnieliśmy wcześniej, w wielu przypadkach obiekt przenikający przez inny obiekt, taki jak przedmiot penetrujący ścianę lub podłogę, można po prostu przesunąć tak, aby dotykał tylko ściany lub podłogi. Inne przypadki mogą być bardziej skomplikowane, a algorytmy iteracyjne są używane do rozwiązania penetracji. Na przykład, jeśli wykryta zostanie penetracja, symulacja może powrócić do poprzedniego kroku czasowego i wykonać krótki krok, aby sprawdzić, czy penetracja nadal występuje. Jeśli nie, symulacja przebiega; w przeciwnym razie symulacja zajmuje jeszcze mniejszy czas. Proces ten powtarza się, dopóki penetracja nie nastąpi. To działa dobrze w wielu przypadkach; jednak czasami penetracja nigdy nie zostanie rozwiązana i symulacja może zostać zablokowana przy mniejszych i krótszych odstępach czasu. Ten brak rozwiązania może zostać przypisany obiektom znajdującym się poza tolerancją odległości zderzenia w jednym momencie, a ze względu na błędy numeryczne wymagane są wyjątkowo małe czasy, aby pozostać poza tolerancją odległości. Niektórzy programiści po prostu umieszczają w kodzie ograniczenie liczby powtórzeń, aby zapobiec utknięciu symulacji, ale konsekwencje w każdej sytuacji mogą być nieprzewidywalne. Opisane powyżej podejście do ciągłego wykrywania kolizji pozwala uniknąć tego rodzaju problemu, przewidując przyszłą kolizję lub penetrację i radzenie sobie z nim z wyprzedzeniem. Bez względu na to, jakie podejście zostanie zastosowane, nastąpi wymiana informacji i wymiana danych między systemami wykrywania kolizji i reagowania, aby uniknąć nadmiernej penetracji. Dodatkowo zdarzają się sytuacje, w których obiekt może spocząć w kontakcie z innym - na przykład pudło spoczywające na podłodze. Istnieje wiele sposobów radzenia sobie z takimi sytuacjami kontaktowymi, z których jedną jest po prostu umożliwienie poradzenia sobie z impulsem. Działa to dobrze w wielu przypadkach; jednak czasami przedmioty w styku spoczynkowym będą drgać z podejściem impulsem-pędem. Jednym z rozwiązań tego roztrzęsionego problemu jest uśpienie tych obiektów - to znaczy, jeśli zostaną zderzone, ale ich względne prędkości są mniejsze niż pewna tolerancja, zostają uśpione. Powiązane, ale nieco bardziej skomplikowane podejście polega na obliczeniu normalnego kontaktu między obiektem a podłogą i ustawieniu tej prędkości na 0. To służy jako ograniczenie, zapobiegając przedostawaniu się przedmiotu do podłogi, jednocześnie umożliwiając jej przesuwanie się po podłodze.

Efekty Siły

Efekty siły działają bezpośrednio lub pośrednio na obiekty w twoich symulacjach. Twój silnik fizyki może zawierać kilka. Na przykład, jeśli twój silnik pozwala użytkownikom poruszać obiektami za pomocą myszy, potrzebna jest pewna wirtualizacja siły przykładanej przez użytkownika za pomocą myszy lub palca na ekranie dotykowym. To jest przykład bezpośredniej siły. Innym efektorem siły bezpośredniej

może być wirtualny silnik odrzutowy. Jeśli skojarzysz ten wirtualny silnik, który wytwarza pewną siłę ciągu, z jakimś obiektem, wówczas powiązany obiekt będzie zachowywał się tak, jakby był popychany przez strumień. Niektóre przykłady pośrednich efektorów siły obejmują grawitację i wiatr. Grawitacja wywiera siłę na obiekty ze względu na ich masę, ale zazwyczaj jest modelowana jako przyspieszenie ciała, a nie jako wyraźna siła. Wiatr może być postrzegany jako wywierający siłę nacisku na obiekt, a siła ta będzie funkcją wielkości obiektu i współczynników oporu. Możecie sobie wyobrazić różnego rodzaju efektory siły, od tych podobnych do tych, które opisałem, prawdopodobnie do tych z innych światów. Cokolwiek sobie wyobrażasz, musisz pamiętać, że siła ma wielkość, kierunek i jakiś centralny punkt zastosowania. Jeśli umieścisz silnik odrzutowy na boku pudełka, skrzynia nie tylko przetłumaczy, ale również się obróci. Wiatr wytwarza siłę, która ma środek ciśnienia, który jest punktem, przez który można przyjąć, że działa całkowita siła wiatru. Kierunek siły i punktu zastosowania są ważne dla przechwytywania zarówno obrotu, jak i obrotu. Weźmy na przykład poduszkowiec, który wymodelowaliśmy w Części 9, który zawierał dwa stery strumieniowe do sterowania i śmigło do ruchu w przód. Każdy z tych efektorów siły bezpośredniej - silniki dziobowe i śmigło - jest stosowany w określonych miejscach poduszkowca. Silniki dziobowe są skierowane w stronę dziobu i skierowane na boki w celu wytworzenia spinu, co pozwala na pewne sterowanie. Śmigło znajduje się na środkowej linii poduszkowca, która przechodzi przez środek ciężkości poduszkowca, dzięki czemu nie tworzy spinu, a zamiast tego popycha jednostkę do przodu. W tym modelu-aerodynamicznym oporze występuje inny efektor siły, który jest pośrednim efektem siły. Siła oporu jest przykładana w punkcie znajdującym się za środkiem ciężkości, tak że tworzy moment obrotowy lub moment, który w tym modelu pomaga utrzymać poduszkowiec prosto; zapewnia pewną stabilność kierunkową. Niezależnie od tego, jakie efekty wywołają siły, wszystkie muszą zostać zebrane dla każdego obiektu i uwzględnione w integratorze numerycznym. W związku z tym twój integrator musi mieć pewne środki dostępu do wszystkich informacji o efektorach siły wymaganych do dokładnej symulacji ich wpływu na każdy powiązany obiekt.

Integrator numeryczny

Integrator jest odpowiedzialny za rozwiązywanie równań ruchu dla każdego obiektu. Pokazaliśmy ci, jak to zrobić wcześniej, w części 7 do części 13. W twoim generycznym silniku fizyki będziesz sprawdzał wszystkie obiekty w symulacji, aby obliczyć ich nowe prędkości, pozycje i orientacje na każdym kroku. Aby wykonać te obliczenia, twój integrator musi mieć dostęp do efektorów siły powiązanych z każdym obiektem. Siły zostaną wykorzystane do obliczenia przyspieszeń, które następnie zostaną zintegrowane do obliczenia prędkości, a te z kolei zostaną wykorzystane do obliczenia pozycji i orientacji. Możesz obsłużyć agregowanie sił na kilka sposobów. Możesz zagregować wszystkie siły na wszystkich obiektach przed zapętleniem obiektów integrujących równania ruchu, ale wymagałoby to dwukrotnego przepętlenia wszystkich obiektów. Alternatywnie, ponieważ przeglądasz listę obiektów w celu zintegrowania równań ruchu każdego obiektu, możesz po prostu agregować siły każdego obiektu podczas etapu integracji. Komplikacja powstaje, gdy pary obiektów nakładają na siebie siły. Liniowa sprężyna i amortyzator, na przykład połączone między dwoma obiektami, przykładają równe i przeciwne siły do każdego obiektu. Siła jest funkcją względnej odległości między obiektami (element sprężyny) i ich prędkością względną (elementem tłumiącym). Tak więc, jeśli podczas danego kroku czasu agregujesz siły na jednym z obiektów w parze, wynikowa siła będzie funkcją bieżącej względnej pozycji i prędkości obiektów. Integracja tego obiektu da mu nową pozycję i prędkość. Następnie, gdy dojdziesz do drugiego obiektu w parze, jeśli wyliczysz siłę sprężyny, będzie to funkcja nowej względnej pozycji i prędkości między obiektami, która obejmuje nowe przemieszczenie i prędkość wcześniej zaktualizowanego obiektu, ale stare przemieszczanie i prędkość bieżącego obiektu. Jest to niespójne z prawem Newtona o równych i przeciwstawnych siłach. Możesz rozwiązać ten problem, przechowując

siłę obliczoną dla pierwszego obiektu i stosując ją do drugiej, bez ponownego obliczania siły sprężyny dla drugiego obiektu.