XIV. Silniki generujące grafikę

Jeśli nic nie poruszyło się w grze, ta gra byłaby bardzo nudna. Oczywiście wiesz już, jak poruszać przedmiotami w grze za pomocą podstawowych elementów fizyki i odrobiny sztuczek z kamer. Wiesz, jak rzucać rakietami, a nawet wyrównać kierunek rakiety z nachyleniem jej ruchu. Ale chociaż twoja rakieta może się poruszyć, nie ma w niej ognia, bo po prostu znika, gdy zderza się z postacią. Twoja postać może podążać za zasadami fizyki, odbijać się i lądować w wodzie, ale tonie w wodzie, nie robiąc nawet plusku. Są to rodzaje małych, ale niezwykle ważnych elementów, które czynią różnicę między kolejnym silnikiem 3D labiryntu a silnikiem; są tym, co daje iskrę w grze. Ostatecznie, im więcej efektów specjalnych możesz włożyć do swojej gry, tym bardziej realistyczny będzie wyglądał i tym więcej punktów bonusowych otrzymasz. Należy jednak pamiętać, że "realistyczny" w tym przypadku nie oznacza fizycznie realistycznego (to znaczy realistycznego pod względem naszego świata). Wręcz przeciwnie, skręcanie praw fizyki zwykle powoduje wspaniałe efekty. Fakt, że postać może wspinać się po ścianach, lub że świat, który tworzysz, ma dziwaczne zielone spadające z nieba, jest najlepszy - o ile nie zakłóca prawidłowej gry. Ponieważ tego typu elementy są generalnie dynamiczne - to znaczy, przychodzą i odchodzą w zależności od kontekstu gry - wymagają struktury generowania grafiki lub, jeśli wolisz, fabryki cząstek. O tym jest ta część. W szczególności ta część omawia następujące kwestie:

■ Naklejki

■ Billboardy

Naklejka

Weź AK-47, postrzelaj kilka rund w kawałek drewna i co się stanie? Jest szansa, że każdy, kto chowa się za kawałkiem drewna, zostanie trafiony pociskiem lub dwoma, a drewno samo rozpadnie się na wiele kawałków. Podobnie, gdy postać w twojej grze strzela dookoła świata, powinien być w stanie zobaczyć obrażenia, które zrobił. Przez większość czasu świat jest uważany za statyczny ze względów optymalizacyjnych. Dzieje się tak dlatego, że im bardziej dynamiczny jest świat, tym wolniejsza gra z powodu problemów z partycjonowaniem / kolizją. Po prostu nie możesz wykonać wstępnej kalkulacji wielu rzeczy w statycznym świecie. Z tego powodu chcesz zmodyfikować świat tylko minimalnie, tak, aby prędkość była minimalnie ograniczona, ale realizm - i ten czynnik chłodu - jest ciągle zwiększany. Obiekt nazywany kalkomanią jest często używany w grach do reprezentowania zmian w materiale świata. Kalkomania jest w efekcie wielobokiem maskującym, który jest nakładany na dotknięty obszar może być zaprojektowany tak, by przypominał dziurę wystrzeloną w ścianie, plamę krwi, ślady poparzeń bomby lub ślad na śnieżnej górze. Nie może wygenerować efektu przezroczystości, ponieważ jest tylko maską, ale może generować "ślady zniszczenia", jak wszyscy lubimy je nazywać.

Problemy z renderowaniem naklejek

Najtrudniejsze jest użycie naklejek - czyli umieszczenie wieloboku maskującego zawierającego obraz, powiedzmy, dziury po pocisku - określa miejsce, w którym należy umieścić wielokąt. Najbardziej oczywistym rozwiązaniem jest wzięcie płaszczyzny, na której ma zostać umieszczona naklejka (na przykład ściana) i naniesienie wieloboku naklejonego na tę część ściany. Może to jednak spowodować problemy, ponieważ ostatecznie dwa poligony są umieszczane w tym samym miejscu. Jaka jest karta graficzna, aby wiedzieć, który z nich powinien być na górze? Możesz wyłączyć buforowanie z, ale w jaki sposób zajmiesz się widocznością? Rozwiązanie jest dość proste. Jeśli przesuniesz obiekt do przodu (w kierunku normalnego wektora, poza ścianą) nieznacznie, możesz oddać kalkomanię bez większych zmian.

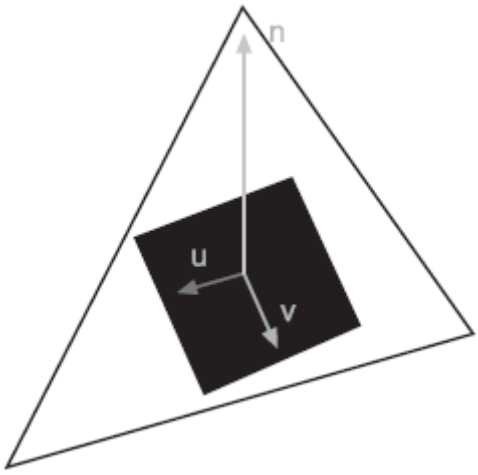
Uwaga

Napotkałem niektóre z najbardziej skomplikowanych i niejasnych rozwiązań tego problemu, w tym obliczenia nowej matrycy projekcyjnej, która przesuwa etykietę do przodu, nie dając wrażenia, że posunęła się naprzód. Jest to jednak doskonały przykład myślenia nieszablonowego, ponieważ eliminuje problem z buforowaniem z, dając jednocześnie taki sam wygląd kalkulatora, obliczając nową matrycę projekcji dla obiektu. To wymaga jednak dużej ilości matematyki, co czyni to rozwiązanie nieatrakcyjnym.

OpenGL i Direct3D są wyposażone w coś, co nazywają offsetem głębokości. Chodzi o to, że możesz wywołać funkcję, która zrównoważy wartości głębokości. Specyficzne funkcje dla OpenGL i Direct3D to odpowiednio glPolygonOffset (1, Bias) i --SetRenderState (D3DRS_DEPTHBIAS, Bias), gdzie Bias jest przesunięciem, które chcesz zastosować. Zwykle wystarcza wartość 1, ale jeśli masz wiele naklejek, możesz chcieć zwiększyć odchylenie. Rozwiązanie jest zatem dość proste. Wystarczy przesunąć bufor Z tak, aby zagwarantować, że wielokąt zawsze przejdzie test z-bufora.

Generowanie naklejki

Teraz, gdy już ustawiłeś naklejkę, aby usiadła nad obiektem za nią (czy to ścianą, podłogą, czy jakimkolwiek innym obiektem), powinieneś być w stanie łatwo obliczyć punkt kolizji między obiektem a wystrzelonym pociskiem, ale w tym momencie musisz jeszcze poprawnie ustawić teksturę. Zacznijmy od uproszczonego przypadku, zakładając, że kolizja występuje tylko z jednym pojedynczym trójkątem. Innymi słowy, twój pocisk jest nieskończenie mały i uderzy tylko w jeden trójkąt w dowolnym momencie. Dalej, musisz wygenerować kalkomanię dopasowaną do tego trójkąta. Niech p będzie punktem kolizji lub środkiem kalkomanii, jeśli wolisz. Nie zawsze konieczne jest zorientowanie kalkomanii, ale w niektórych przypadkach - jako przykład można podać ślad śniegu - przydatne może być ustawienie naklejki. W takim przypadku należy ustawić kierunek etykiety, jak pokazano na rysunku 14.1,



orientacja może na przykład odpowiadać kierunkowi, w którym chodzi postać. Powinieneś również mieć wektor normalny dla trójkąta; u definiuje obrót naklejki wokół normalnego wektora. Jeśli nie zależy ci na kierunku, możesz zbudować drugi wektor za pomocą procesu ortogonalizacji Gram-Schmidta za pomocą n, lub możesz wygenerować własny wektor wybierając punkt a z trójkąta generującego wektor ap = a? str. Jeśli zdecydujesz się na własny, musisz upewnić się, że u jest ortogonalne do n, lub Twoja kalkulacja nie będzie poprawnie wyrównana. Jeśli twoja postać idzie w tym samym kierunku i chcesz dodać swój ślad, to masz kierunek postaci d. d niekoniecznie wyrównany z powierzchnią, na której chadza postać. Na przykład, przypuśćmy, że postać wspina się na wzgórze. W

takim przypadku może on iść prosto, ale ze względu na wzgórze jego kierunek jest naprawdę prosty i skierowany w górę. Z tego powodu wektor orientacji u musi być wyrównany z powierzchnią, inaczej kalkomania zanurzy się w obiekcie znajdującym się za nim. Możesz łatwo rozwiązać ten problem, obliczając rzut wektora na płaszczyznę. Dokładniej, jest to prostopadły komponent rzutu u na n. Na koniec zdefiniuj v jako wektor, który uzupełnia podstawę, a tym samym jest prostopadły do u i n. Czy to ja, czy to brzmi jak produkt krzyżowy? Jeśli przeanalizujesz rysunek 14.1, zobaczysz, że masz już odpowiedź, której szukasz. Aby znaleźć punkty końcowe powierzchni, u i v są kluczami. Najpierw musisz znormalizować, a następnie pomnożyć połowę ich szerokości / wysokości (połowa z każdej strony od punktu centralnego daje pełną długość). Następnie możesz uzyskać cztery rogi kalkomanii, dodając wektory z punktu środkowego i zmieniając znaki. W ten sposób cztery rogi zdefiniowane zostałyby przez następujące wstępnie obliczone czynniki:

$$\mathbf{u}' = \frac{Height \cdot \mathbf{u}}{2\|\mathbf{u}\|_2}$$

$$\mathbf{v}' = \frac{Width \cdot \mathbf{v}}{2\|\mathbf{v}\|_2}$$

i określone przez następujące

$$\mathbf{TopLeft} = \mathbf{p} - \mathbf{u} - \mathbf{v}$$

$$\mathbf{TopRight} = \mathbf{p} - \mathbf{u} + \mathbf{v}$$

$$\mathbf{LowerLeft} = \mathbf{p} + \mathbf{u} - \mathbf{v}$$

$$\mathbf{LowerRight} = \mathbf{p} + \mathbf{u} + \mathbf{v}$$

Przycinanie Trójkątów

Teraz, gdy wiesz, gdzie renderować kalkomanię, czas spojrzeć na wycinek i nie tylko. Na przykład, co się stanie, jeśli twoja postać wystrzeli pocisk na krawędzi ściany? Zgodnie z przyjętym do tej pory modelem, kalkomanie będą widoczne na krawędzi ściany, z częścią nalepki na ścianie, a jej część poza ścianą. Innymi słowy, kalkomania będzie wyglądać tak, jakby ktoś utknął na krawędzi kawałka papieru zawierającego obraz dziury po kuli, z częścią papieru na ścianie i częścią zwisającą w powietrzu. Aby rozwiązać ten problem, należy przyciąć kalkomanię tak, aby pojawił się tylko fragment, który powinien pojawić się na ścianie, podłodze lub innej powierzchni. Aby użyć obcinania, należy najpierw założyć, że świat gry składa się z trójkątów, ponieważ w końcu tak wszystkie nawiasy są konwertowane. Tak więc patrzysz na obcinanie trójkąta na krawędzi ściany. Aby ułatwić sobie wizualizację, umieść karteczkę na krawędzi drzwi. Jeśli przykleisz karteczkę wystarczająco blisko krawędzi drzwi, prawdopodobnie wystaje. Innymi słowy, karteczka nie jest całkowicie na drzwiach; jego część wystaje w powietrze. Właśnie to chciałbyś spiąć. Pomysł na obcinanie opiera się na przecięciu linii (krawędzi drzwi) i trójkąta (karteczkę lub kalkomanię). Matematycznie mówiąc, gdyby twoje systemy były w 100 procentach precyzyjne, wykonaliby to obliczenie bez problemu; Niestety tak nie jest. Linia jest na tej samej płaszczyźnie, co trójkąt kalkomanii, co oznacza, że wystąpią problemy precyzji Twoje obliczenia są bardzo trudne w 3D. Zamiast tego, musisz zastąpić krawędź płaszczyzną, która przecina trójkąt o tym samym kącie co krawędź.

Istnieje kilka płaszczyzn, które możesz wybrać. Jedyna zasada mówi, że samolot musi przeciąć trójkąt w taki sam sposób, jak to zrobiła krawędź. W ten sposób można wygenerować płaszczyznę z wektorem krawędzi, normalnym wektorem trójkąta i jednym punktem krawędzi. Innym ważnym wyborem jest wybór płaszczyzny opisanej przez jeden z trójkątów, które mają tę samą krawędź co wybrana krawędź. Na przykład, wracając do przykładu drzwi, możesz wybrać powierzchnię drzwi prostopadłą do powierzchni, na której przykleiła się lepka kartka. To rzeczywiście przeciąłby lepką nutę we właściwej pozycji. Wreszcie, nie możesz pracować nad trójkątem jako całością. Zamiast tego należy przyjrzeć się serii mniejszych problemów, z których pierwszy polega na znalezieniu przecięcia między segmentem linii a płaszczyzną. Tak więc, biorąc pod uwagę trójkąt, będziesz musiał spojrzeć na wszystkie trzy jego krawędzie i przetestować je względem płaszczyzny. Ogólnie rzecz biorąc, algorytm jest następujący. Najpierw musisz wybrać jedną krawędź z trójkąta kalkomanii. Następnie, dla każdej krawędzi z punktami końcowymi p_1 i p_2, wykonaj następujące czynności:

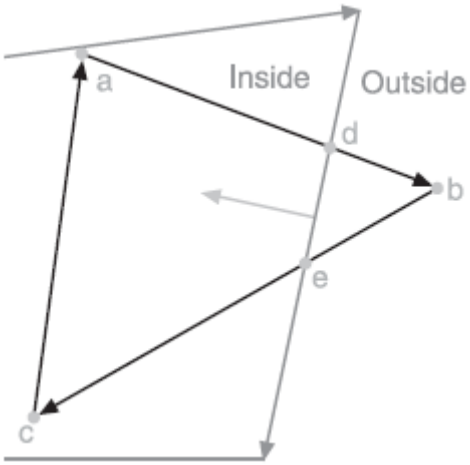
■ Jeśli oba punkty leżą po dodatniej stronie płaszczyzny, dodaj ostatni punkt (p_2).

■ Jeśli oba punkty leżą po ujemnej stronie samolotu, nie rób nic.

■ Jeśli przejdziesz od środka (p_1) do zewnątrz (p_1), musisz dodać punkt przecięcia płaszczyzny z linią.

■ Jeśli przejdziesz z zewnątrz (p_1) do środka (p_1), musisz dodać punkt przecięcia między płaszczyzną i linia i musisz dodać ostatni punkt (p_2).

Za pomocą tego algorytmu zaczynasz od pustej listy wierzchołków i wstawiasz je zgodnie z zaleceniami algorytmu. Krok dla jednej krawędzi tła z kalkomanią pokazano na rysunku 14.2. Możesz otrzymać więcej wierzchołków niż wymaga tego trójkąt. W takim przypadku możesz nadal renderować kształt, ale jako wielokąt zamiast trójkąta lub, jeśli wolisz, możesz mozaikować wielokąt (tzn. rozłożyć wielokąt na zestaw trójkątów), po prostu wybierając punkt, z którego zostaną utworzone wszystkie krawędzie. Na rysunku 14.2,



zakładając, że zaczynasz od punktu a, zaklasyfikowałeś punkt a jako wewnątrz, stosując zasady omówione w częścu 4 (to znaczy, zastępując punkt wewnątrz równania normalnego i sprawdzając znak). Oczywiście, c miałby taki sam wynik, podczas gdy b byłby oznaczony jako zewnętrzny. Ponieważ na początku przeszliście od środka do środka (od a do b), dodaliśmy skrzyżowanie d. Następny punkt to c; sprawdzając poprzedni punkt b, zauważysz, że teraz przechodzisz z zewnątrz na stronę, co oznacza, że musisz dodać kolejny wierzchołek e, a także musisz dodać c. Na koniec, dla ostatniej krawędzi, przechodzisz od c do a; ponieważ oba są w środku, dodajesz a. Już widziałeś, jak obliczyć punkt przecięcia się płaszczyzny z linią w części 4, więc jedyną rzeczą, której już nie ma, jest metoda obliczania samego samolotu ze względu na krawędź trójkąta. Na szczęście jest to łatwe. Jeśli weźmiesz

krzyżówkę normalnej kalkomanii z wektorem wygenerowanym przez krawędź trójkąta, otrzymasz normalny kierunek. Ponieważ płaszczyzna <n, D> również ma przesunięcie D, wystarczy podstawić jeden z punktów krawędzi tła, aby obliczyć 4-wektor dla płaszczyzny. W ujęciu matematycznym: krzyżowy produkt jest dość wybredny w odniesieniu do kolejności wektorów, dlatego ważne jest, aby zauważyć, że ta formuła działa tylko wtedy, gdy wielokąty są zdefiniowane zgodnie z ruchem wskazówek zegara i jeśli normalny wektor wskazuje na kamerę. Jeśli nie, musisz odwrócić kolejność produktów krzyżowych, aby uzyskać normalny wektor w przeciwnym kierunku. (Jeśli normalny nie jest skierowany w stronę kamery, to dlaczego zaczynasz renderowanie wielokąta?) Reszta powinna być bułka z masłem. Kilka porad dotyczących kalkomanii: Upewnij się, że sprawdzisz przecięcie naklejek z otaczającymi wielokątami (nie tylko przecinającym się trójkątem), ponieważ naklejka może w rzeczywistości obejmować dwa trójkąty. Na koniec, pamiętaj, aby poprawnie poprawić współrzędne tekstury podczas cięcia trójkąta na kawałki. Współrzędne tekstury powinny być interpolowane dokładnie w ten sam sposób, w jaki interpolujesz wierzchołki z jednego punktu do drugiego. Jedyna różnica polega na tym, że zamiast interpolować <x, y, z>, robisz to dla <u, v>. Przypomnijmy, że obliczanie przecięcia płaszczyzny i linii obejmuje parametr t. Ta wartość, jeśli wektory są znormalizowane, reprezentuje odległość od punktu początkowego do punktu końcowego. W ten sposób można interpolować współrzędne tekstury z jednej pozycji do drugiej, używając tego samego t. Równanie będzie wyglądać jak a + tb, gdzie a jest pierwszym punktem tekstury, a b drugim dla krawędzi (system liniowy, który powinieneś już wiedzieć, jak rozwiązać). Na szczęście ten proces nie musi być stosowany do każdej pojedynczej klatki. Możesz po prostu obliczyć to tak, jak to się dzieje i zachować wierzchołki podczas podróży. Większość gier ogranicza liczbę naklejek, które można wkleić na ekranie; zwykle używają kolejki, w której nowsze naklejki wymazują starsze. Ponadto, zwykle mają one zanikają z czasem, po prostu zwiększając jej przezroczystość do punktu, w którym nie jest już widoczny, a następnie zostaje usunięty. Jeśli chcesz być bardzo wymyślny, możesz użyć koncepcji o nazwie bumpmapping, aby te kalkulacje wyglądały jeszcze bardziej realistycznie; dowiesz się więcej o bumpmapping w kolejnych rozdziałach. Inną interesującą sztuczką jest trwałe wpłynięcie na teksturę poprzez aktualizację samej tekstury. Jeśli na przykład strzeliłeś do muru i wiesz, że ta dziura po kuli musi tam pozostać przez cały czas trwania gry, możesz zastosować kalkomanię do samej tekstury. Innymi słowy, oblicz nową teksturę, renderując teksturę i renderując kalkomanię nad nią, a następnie przechowując tę teksturę. Jedynym haczykiem jest to, że nie możesz tego zrobić dla każdej dziury po kuli, bo w przeciwnym razie otrzymasz więcej tekstur, niż możesz przypisać. Powiedział, że przyspiesza grę, jeśli masz na to pamięć i jeśli możesz zagwarantować, że musi tam pozostać.

Billboardy

Czy zastanawiałeś się kiedykolwiek, jakie są fajne efekty, takie jak rozbłyski soczewkowe, rozbryzgi wody, eksplozje krwi, czy dym i ogień płonący za rakietą? Wszystko to dzieje się dzięki billboardom. Billboardy działają w sposób podobny do księżyca. Czy zauważyłeś, że księżyc nigdy się nie zmienia? Rozumiem przez to, że bez względu na to, która strona księżyca jest zapalona, zawsze widzisz te same dziury, uderzenia i tym podobne - nawet jeśli księżyc jest kulą, która nie tylko obraca się wokół Ziemi, ale także wokół siebie. Dzieje się tak dlatego, że księżyc obraca się w taki sposób, że z punktu widzenia Ziemi ta sama strona jest zawsze wyświetlana. Billboardy wykorzystują tę samą zasadę do generowania zaskakująco oszałamiających efektów z punktu widzenia gracza. Gdybyś miał modelować dym, ogień, wodę lub jakikolwiek inny rodzaj cząstek za pomocą opisu 3D, potrzeba czasu na renderowanie każdego fragmentu układanki. Wyobraź sobie, że musisz oddać dym za rakietą za pomocą powtarzających się i animowanych modeli 3D. Karta wideo z pewnością umrze. Zamiast tego, możesz oddać dym raz w teksturach do wykorzystania jako billboard i pokazać sprite'a zamiast pełnoprawnego modelu 3D, kiedy chcesz go narysować. Jest to wyraźnie preferowane w odniesieniu do prędkości, ale oczywiście jakość nie jest taka sama, ponieważ widać tylko jedną stronę. Jeśli chodzi o efekty

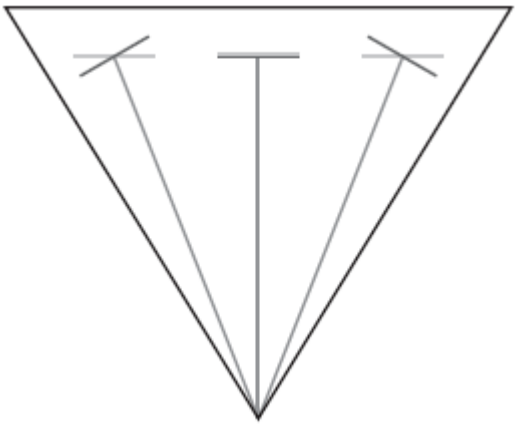
środowiskowe, takie jak dym, ogień i woda, jest to doskonałe, ponieważ generalnie nie mają one bardzo złożonej głębokości - jeśli w ogóle.

Wydajne, nieskrępowane billboardy

Jedynym problemem z billboardami jest to, że widzisz tylko jedną stronę obiektu wyświetlaną na billboardzie. Wracając do Doom I / II dni, silniki 3D wykorzystywały billboardy do renderowania złych ludzi. W pewnym momencie John Carmack (sława oprogramowania ID) doszedł do wniosku, że komputery PC są wystarczająco szybkie, by tolerować prawdziwe obiekty 3D, a Quake ożył oszałamiającymi postaciami (w porównaniu do Dooma). W niektórych okolicznościach, na przykład gdy są bardzo daleko, jest to nadal najlepszy sposób. Jak to działa? Na początek musisz wziąć pod uwagę każdy obiekt jako cząstkę, jeden unikalny punkt w przestrzeni. Możesz zastosować fizykę lub dowolną dowolną formułę ruchu do tych punktów. Sam billboard musi być jak księżyc, pokazując tylko jedną stronę obiektu. Aby rozwiązać ten problem, możesz użyć sztuczki podobnej do tej, której użyłeś do naklejek. Biorąc pod uwagę punkt środkowy cząstki, znajdź dwa przesunięte wektory, które skonstruują cząstkę taką, że normalny wektor billboardu jest dopasowany do tego w kamerze. Kluczowa jest tutaj macierz transformacji modelu. Macierz transformacji modelu jest zdefiniowana jako macierz, która przekształca obiekty w współrzędne kamery. Jako taki obejmuje matrycę obiektu, matrycę światową i matrycę kamery w jednym. Jeśli macie macierz transformacji modelu tożsamości, kamera będzie skierowana w dół w kierunku osi Z. W ten sposób możesz utworzyć swój billboard w przestrzeni przed zastosowaniem matrycy, przekształcić współrzędne, a następnie renderować normalnie. Billboard renderowany na płaszczyźnie xy jest zawsze normalny dla kamery przed transformacją, więc jeśli przekształcisz ten billboard przy użyciu tej samej matrycy, koniecznie otrzymasz billboard, dla którego normalna kamera po transformacji jest wyrównana do billboardu . Aby przyspieszyć, możesz przekształcić dwa rogi billboardu (możesz również wykonać dwa prostopadłe kierunki). Na przykład możesz zmienić lewy dolny i prawy dolny róg, przekształcając dwa wektory <Szerokość / 2, Wysokość / 2, 0> i <-Szerokość / 2, Wysokość / 2, 0>. Są to dwa wektory, które mogą pomóc w wygenerowaniu czterech rogów po transformacji, jak pokazano wcześniej. W związku z tym nie trzeba przekształcać czterech rogów, ponieważ pozostałe dwa rogi są liniową kombinacją dwóch pozostałych; mają tylko inwersję znaku. Po transformacji tych dwóch wektorów, piękno tej techniki jest takie jakie możesz zachować do przesunięcia i będzie ono takie samo dla każdej pojedynczej cząsteczki, którą możesz wywietrzyć. Aby podsumować, zacznij od transformacji wektorów u = <szerokość / 2, wysokość / 2, 0, 1> i v = <-Szerokość / 2, Wysokość / 2, 0, 1> z macierzą transformacji modelu. Następnie, dla każdej cząstki, musisz narysować kwadrat (<p + u, p + v, p- Y, p- V>).

Precyzyjny nieograniczony billboard

Poprzednia metoda jest świetna, ponieważ jest bardzo wydajna. Zakładając, że offset musi zostać obliczony tylko raz, pozostałe operacje są jedynie dodatkami i odejściem. Niestety nie jest to najdokładniejsza metoda. Jeśli myślisz o aparacie jako o ciele, a nie o rzeczywistym wektorze, jasne jest, że billboardy pośrodku kamery powinny być skierowane bezpośrednio w stronę aparatu. Z drugiej strony billboardy na krawędzi stożka ściętego powinny w rzeczywistości być normalne w kierunku wskazywanym przez ścięty stożek (czyli wektor generowany przez środek cząstki i położenie kamery), jak pokazano na rysunku 14.3.



Problem z tą wersją polega na tym, że jest zależna od położenia, co oznacza, że przesunięcie będzie musiało zostać obliczone za każdym razem, gdy chcesz renderować nową cząsteczkę. Taka jest cena za większą dokładność. Zaczynasz od skonstruowania wektora, który przechodzi z kamery na billboard (Billboard.Position - Cam.Position), ponieważ definicja tej techniki wymaga że płaszczyzna billboardu jest ortogonalna do tego samego wektora. Niech q będzie tym wektorem i dodatkowo zdefiniuj wektor r jako wektor normalny, którego użyłeś w poprzedniej metodzie. Wektor ten można uzyskać przez przekształcenie wektora <0, 0, 1> w macierz transformacji. Kiedy masz te dwa wektory, rozwiązanie polega jedynie przy iloczynie wektorowym dla przesunięcia wysokości i jeszcze innym iloczynie wektorowym dla przesunięcia szerokości:

$$\mathbf{x} = \frac{\mathbf{r} \times \mathbf{q}}{\left\| \mathbf{r} \times \mathbf{q} \right\|_2}$$

$$\mathbf{y} = \mathbf{q} \times \mathbf{x}$$

zrozumienie wektorów i ich operacji jest kluczem do zrozumienia tego. Wszystko, co tutaj robisz, to wektory kompilacji, które są ortogonalne do dwóch wektorów, które masz. Gdzie x i y są wektorami przesunięcia, możesz zbudować dwa narożne wektora przesunięcia, obliczając x + y i x - y, i używając tej samej techniki, którą wymieniono wcześniej, aby obliczyć cztery rogi. Przesunięcia w tym przypadku są tylko wektorami, które doprowadzą cię do najbliższej krawędzi (jednej z boków lub górnej lub dolnej); w związku z tym musisz dodać lub odjąć je, aby przejść przez róg. Teraz wszystko, co musisz zrobić, to zdecydować, która technika jest odpowiednia dla twoich potrzeb.

Ograniczone billboardy

Nieograniczone quady to billboardy, które mogą poruszać się swobodnie w kosmosie. Niezależnie od tego, w którą stronę patrzy kamera, znajdują się tam nieskrępowane quady, patrząc prosto w oczy. Chociaż nieskrępowane quady mogą być przydatne do wielu rzeczy, nie obejmują wszystkich rodzajów możliwych billboardów. Na przykład rozważ pochodnię. W takim przypadku może być praktyczne zdefiniowanie ognia tak, aby zmieniał kierunek podczas obracania się wokół pochodni, ale nie wtedy, gdy patrzysz na latarkę z góry. W rzeczywistości może się przydać wyświetlenie jednego billboardu, gdy patrzysz na latarkę z góry, a na drugą, gdy patrzysz na nią z boku. Tego typu billboardy są specjalne, ponieważ są zablokowane na osi. Pochodnia boczna jest na przykład zablokowana względem osi Y, która porusza się w górę iw dół.

Uwaga

Chociaż równanie zostanie wyprowadzone na billboardy obracające się wokół osi Y, można łatwo zastosować tę samą logikę, aby znaleźć równanie dla obrotu wokół dowolnej innej osi.

Pierwszą rzeczą, którą należy zauważyć, to to, że wysokość billboardu nigdy się nie zmienia, niezależnie od kierunku. Ponieważ billboard tylko obraca się wokół osi y, jego komponent y nie porusza się. W konsekwencji przesunięcie tablicy w y można łatwo wyrazić jako połowę wysokości:

$$\mathbf{y} = \left\langle 0, \frac{h}{2}, 0 \right\rangle$$

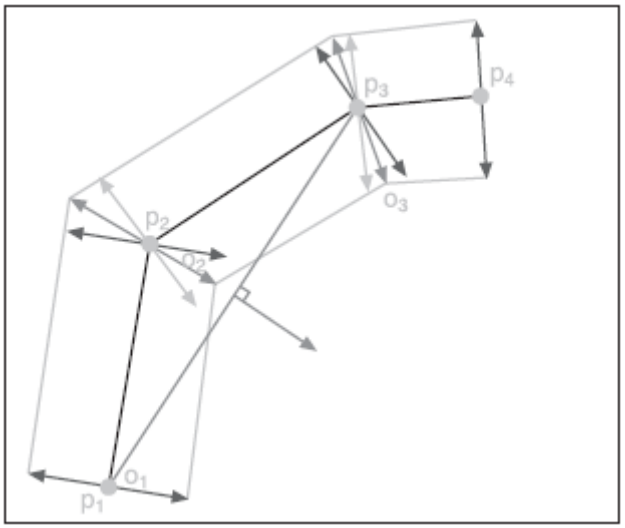
To było zbyt łatwe; a co ze współrzędnymi x i z? Wiesz, że płaszczyzna billboardu musi być równoległa do wektora kamery i masz już wektor, który jest wyrównany z płaszczyzną <0, 0, 1>, co wymusza wyrównanie tablicy z osią y. Obliczyć iloczyn krzyżowy tych dwóch wektorów i voila, otrzymasz następujące, gdzie q jest zdefiniowane jako wektor, który przechodzi z kamery na billboard:

$$\mathbf{x} = \frac{w}{2} \cdot \frac{\mathbf{q} \times \langle 0,0,1 \rangle}{\left\| \mathbf{q} \times \langle 0,0,1 \rangle \right\|_2}$$

Ponownie, jeśli chciałbyś użyć mniej precyzyjnej metody, możesz użyć normalnego wektora kamery, który jest uzyskiwany przez macierz transformacji modelu i jest taki sam dla wszystkich billboardów. Wybrana metoda zależy od szybkości i jakości obrazu.

Szok z liniami łamanymi

Kiedy zobaczyłem błyskawicę w Return to Castle Wolfenstein (RTCW), byłem zachwycony jej pięknem i realizmem. Rzeczywiście, błyskawica była używana w grach, ale nigdy nie była tak piękna jak w RTCW. Główną różnicą między błyskawicą w tej grze a błyskawicą, która pojawiła się w innych jest to, że RTCW użyło wspaniałych kolorów i tekstur, aby osiągnąć efekt. Błyskawica może być również renderowana za pomocą formy billboardu, która jest generowana w środowisku wykonawczym. Jedyną czkawką jest budowanie billboardu dopasowanego do kamery i poruszającego się w czasie. Pomysł jest znowu prosty. Jeśli uważasz błyskawicę za zestaw połączonych punktów, które się poruszają, możesz łatwo zobaczyć, jak to działa. (Przy okazji, możesz również użyć tego podejścia do czegoś bardziej ezoterycznego, takiego jak bicz, łańcuch lub inne podobne elementy, które są oparte na łańcuchach). Podstępem jest tutaj zbudowanie czegoś, co zwykle nazywa się polilinią. Polilinia to nic innego jak gruba linia. Możesz spróbować renderować go jako zbiór prostokątów lub grubych linii, ale po prostu nie będzie dobrze wyglądać w złączeniach, ponieważ całkowicie nie będziesz mieć złączeń. Z tego powodu potrzebujesz metody generowania ciągłej, grubej linii, w tym połączeń. Zamiast używać prostokątów lub grubych linii, dlaczego nie korzystać z czworokąta? W ten sposób można wypełnić luki łączące, ponieważ nie musi to być prostokąt. Możesz zbudować taki kształt, obliczając przesunięcie sprzężenia od punktu środkowego, jak pokazano na rysunku 14.4.



Ponownie, to tylko gra z właściwościami wektorów. Potrzebny jest wektor prostopadły do wektora utworzonego przez dwa ciągłe punkty zdefiniowane przez ciąg. Punkt ten musi być prostopadły do kamery i do odcinka linii, w związku z czym stosuje się produkt krzyżowy. Możesz użyć albo precyzyjnej, albo wolniejszej techniki lub szybkiej, ale mniej dokładnej techniki omawianej wcześniej w tym rozdziale. Jeśli wybierzesz szybką technikę, twój wektor kamery będzie stały; jeśli nie, wektor kamery zależy od pozycji punktu, który obecnie oceniasz, czyli q (wektor z kamery do cząstki) we wcześniejszej sekcji. Jeśli weźmiesz pod uwagę bardziej precyzyjny i skomplikowany przypadek (drugi można łatwo uzyskać za pomocą tej samej logiki, co podano wcześniej), musisz zdefiniować wektor d jako wektor, który przechodzi od jednego punktu ciągu do następnego punktu na string.Tak więc przesunięcie o od tego punktu jest zdefiniowane jako takie, gdzie t jest zdefiniowane jako odległość ortogonalna lub, jeśli wolisz, szerokość grubej linii:

$$\mathbf{o} = \frac{t(\mathbf{d}\times\mathbf{q})}{\|\mathbf{d}\times\mathbf{q}\|_2}$$

Dlatego dwa wierzchołki, które chcesz dodać do listy dla bieżącego punktu p, to p + oi p? o. Jest jednak jeden problem, jeśli zrobisz to dla każdego punktu. Załóżmy na przykład, że masz bardzo ciasny obrót o 90 stopni. W takim przypadku przedłużenie cząstki leżałoby dokładnie na następnej krawędzi. Jest to wyraźnie niepożądane. Jeśli spojrzysz jeszcze raz na rysunek 14.4 i uwzględnisz tylko przesunięcia, które są prostopadłe do poprzedniej linii, oczywiste jest, że w niektórych przypadkach deformuje to szerokość linii. W przypadku 90-stopniowej kostki zapada się całkowicie na jednym końcu. Alternatywnie można obliczyć średnią między przesunięciem bieżącej krawędzi a odsunięciem następnej krawędzi. Innym sposobem patrzenia na to jest wygenerowanie wektora d takiego, że jest to wektor przechodzący od punktu p_{i+1} do p_i? 1, jak pokazano na rysunku 14.4. Tak więc, z wyróżnionym wyjątkiem pierwszych i ostatnich wierzchołków, które są obliczane jak pokazano wcześniej, można obliczyć przesunięcie linii łamanej a pomocą następującej metody:

$$\mathbf{d} = \mathbf{p}_{i+1} - \mathbf{p}_{i-1}$$

Aplikacje

Do tej pory koncentrowałeś się na podstawach działania. Teraz możesz rzucić okiem na kilka zgrabnych rzeczy, które możesz zrobić dzięki tym koncepcjom. Prawdopodobnie masz już dobre pojęcie o tym, w jaki sposób te koncepcje wchodzą w grę; ich zastosowania są tylko ograniczone twoją wyobraźnią.

Wybuch

Jaka byłaby gra bez wybuchów? Kiedy napisałem małą grę Lemmings dla TI-86, uśmiechem graczy było to, że replikuje sekwencję samozniszczenia lemingów. Zaczynają puchnąć iw końcu każdy z nich wysadza się w powietrze. Stworzenie eksplozji w 3D jest nieco bardziej zaangażowane niż stworzenie właściwych duszków; obejmuje system cząstek wyposażony w silnik fizyki i popychany przez silnik do wyświetlania billboardów. Ideą eksplozji jest to, że początkowo wszystkie cząstki są skupione wokół tego samego punktu w przestrzeni. Przed wybuchem cząstki otrzymują różne znormalizowane wektory prędkości. Po wybuchu tworzy on chmurę sferyczną (z powodu normalizacji), która spada zgodnie z regułami grawitacji. W rzeczywistości bardzo łatwo jest dokonać eksplozji. Najtrudniej jest uzyskać odpowiednie fizyczne stałe i odpowiednie tekstury, aby billboard wyglądał wystarczająco realistycznie. Jako naprawdę miły akcent, zastanów się nad ustawieniem swojej gry tak, aby eksplozje miały wpływ na ściany na ekranie. W ten sposób, jeśli obliczysz przecięcie cząstki ze światem, możesz wykryć, czy jakakolwiek krew spłynie po ścianie, czy nawet uderzy w inną postać. Ten rodzaj schematu można również zastosować do rozprysków wody, fajerwerków, dymu strzeleckiego i różnych innych rodzajów eksplozji.

SSSSssssssmokin "

A la'Jim Carrey w Masce, kolejnym fajnym efektem jest dym. Jest to całkiem prosty generator cząsteczek do wdrożenia. Wszystko czego potrzebujesz to pozycja obiektu, który generuje dym, i możesz pozwolić obiektowi generować jedną chmurę dymu co x milisekundy. Aby poruszyć dym dookoła świata, możesz ponownie użyć czegoś bardzo podobnego do eksplozji. Bardziej interesujące jest jednak to, że dym znika z czasem. Na przykład, gdy ktoś tworzy obłok dymu z papierosem, rozszerza się do punktu, w którym miesza się całkowicie z otaczającym powietrzem. (Oczywiście, jeśli robisz to zbyt wiele razy w pokoju, kończysz z pokojem, w którym nic nie widzisz, ale masz pomysł.) Aby wziąć to pod uwagę, możesz być bardzo podniecony, mając billboard rozwija się z czasem. Quake używa tej strategii podczas wystrzeliwania rakiet. Wystrzel rakietę w Quake III, a zauważysz, że dym rozszerza się. Ponadto, możesz ustawić billboard, aby zanikać z czasem, tak, aby ostatecznie zmieszał się z tłem. Możesz to łatwo osiągnąć, zmniejszając wartość alfa tablicy reklamowej w czasie, tak jak możesz skonfigurować kalkomanię, aby stopniowo zanikać. (Nazywa się to podaniem obiektu "życie").

Flara

Oto efekt, który, moim zdaniem, nie jest bardzo dobrze zaimplementowany w wielu grach. Flary obiektywu mogą czasami poprawić jakość wizualną gry, ale musisz upewnić się, że nie kradną całej sceny. Rozumiem przez to, że rozbłysk nie powinien być tak oczywisty, że pomniejsza rzeczywistą grę. Niestety, wiele gier ma tendencję do używania ostrych kolorów, co sprawia, że rozbłysk jest dość oczywisty, a przez to mało atrakcyjny. Flara obiektywu jest w rzeczywistości zjawiskiem fizycznym, które występuje tylko wtedy, gdy obiektyw jest używany w kamerze. Istnieją fizyczne reprezentacje, jeśli naprawdę chcesz być poprawny fizycznie, ale w rzeczywistości nie musisz być w dobrej formie, aby błyski wyglądały dobrze - i to jest cały powód, by używać flar. Rozbłysk jest ogólnie postrzegany jako zestaw okręgów i iskier ustawionych w linii prostej, która pokrywa się ze środkiem ekranu i położeniem źródła światła. Tak więc, jeśli słońce znajduje się w określonej pozycji na ekranie, wszystkie kręgi wyrównawcze będą ustawione wzdłuż nieskończonej linii, która obejmuje obszar między środkiem ekranu a słońcem. Musisz się upewnić, że rozróżniasz dwa. (Właściwie mówię tutaj o środku ekranu, a nie o środku świata.) Kiedy tworzysz flarę, używane tekstury zazwyczaj zawierają iskrę, kółko gradientu

(przechodząc od jasnego do ciemnego), inne koło gradientu (przejście z ciemności do światła) i aureola. Możesz komponować te obrazy według własnego uznania. Jedynym ograniczeniem jest to, że należy nadać każdemu obrazowi wartość interpolacji t, która jest ustalona dla linii generowanej między źródłem światła a środkiem ekranu. To zagwarantuje, że flary będą się prawidłowo poruszać. Jeśli środek flary jest zasłonięty przez inny obiekt, nie powinieneś w ogóle rysować flary. Powtórzę jeszcze raz: nie powinieneś przycinać flary; nie powinieneś w ogóle rysować płomienia. Możesz samemu sprawdzić te zjawiska fizyczne. Spójrz na latarnię w nocy i ukryj źródło światła ręką. Jeśli prawie zamkniesz jedno oko, zauważysz, że nie widzisz już błysku światła. To prowadzi nas do kolejnego tematu. To może być bardzo brzydkie, aby zaimplementować flarę wszędzie w twojej grze (lub w ogóle). Chociaż wielu zgodzi się na tę liczbę, wielu zgodziłoby się również, że obiekty takie jak latarnie, na przykład, odniosłyby korzyść z małej zawartej iskry w pozycji źródła światła. Flare nie jest chyba właściwym słowem; być może halo jest bardziej odpowiednie. Jesteś przyzwyczajony do tego, że wokół świateł widzisz odpowiednio duże halo, więc nie jest to zły pomysł, aby dodać je do swoich scen. Z drugiej strony, w grze, niezależnie od tego, czy dodasz halo, może zależeć od warunków środowiskowych. Na przykład, gdy pada deszcz, bardziej widoczne halo staje się widoczne z powodu zwiększonego załamania światła i deszczu. W rzeczywistości, w ciągu dnia, prawdopodobnie nie zauważysz aureoli w ogóle, chyba że jesteś w środku. Idź z tym, co wygląda dobrze. Są one realizowane za pomocą billboardów, więc jedyną prawdziwą pracą, jaką musisz wykonać, jest wybór / stworzenie odpowiednich tekstur.

Ogień

Ogień nie jest łatwy do odtworzenia. Niektóre implementacje mają tendencję do używania ograniczonych billboardów, co działa bardzo dobrze, jeśli stać Cię na zestaw tekstur, które po animacji sprawiają, że obszar wygląda jak poruszający się płomień (tak jak w przypadku Quake'a). Innym podejściem do ognia jest zasymulowanie go wiązką cząstek. Pomysł, aby połączyć kilka drobnych cząstek, z których każda jest po prostu niewielką, częściowo jasną plamą, tak, że kolor przechodzi z ciemnego pomarańczowego na jasny żółty / biały. Aby uczynić te cząstki realistycznymi, potrzebujesz innego generatora cząstek, który generuje cząstki o danym życiu, ponieważ cząstki w końcu muszą zgasnąć. To dokładnie symuluje krawędź płomienia. Trudność polega na uzyskaniu wzoru na ruch cząstek. Jeśli działasz pod rakietą, nie jest to takie trudne, ponieważ grawitacja zajmuje większą część; możesz po prostu pozwolić niższemu przyspieszeniu grawitacyjnemu odciągnąć cząstki. Jeśli tak nie jest, możesz przejść do alternatywnej trasy. Możesz losowo wybrać prędkość cząstek, ale musisz je tak wygenerować, by były generowane ostrym stożkiem. Wykorzystanie cylindrycznej przestrzeni współrzędnej pomogłoby to osiągnąć. W tym przypadku współrzędne y będą losowane, tak jak współrzędne x i z w pierścieniu stożka na wysokości y. Jeśli zostawisz to jako takie, cząsteczki poruszą się tylko w stożku, który nie będzie dobrze wyglądał. Aby wprowadzić przypadkowość, która pojawia się w pożarach, musisz umożliwić cząstkom lekkim wyjście poza stożek. Ostatnim krokiem, który czyni to bardziej realistycznym, jest zwiększenie rozmiaru cząsteczki, gdy znajduje się ona blisko środka stożka. W ten sposób cząstki znajdujące się bliżej źródła powinny mieć więcej energii niż te, które są dalej. Ponieważ energia jest głównie wzdłuż osi Y stożka, to tam rozmiar powinien być największy. Im dalej cząstka jest od tego punktu, tym mniejsza powinna być. Tak więc na wielkość cząstki powinno wpływać jej życie (im starsza cząsteczka, tym mniejsza powinna być) i jej odległość od środka (im bliżej środka, tym większy powinien być). Jeśli grasz liczbami, ostatecznie osiągniesz coś, co wygląda dobrze.