XVII. Wypełnianie luk technikami renderowania

Kilka lat temu, kiedy DOS rządził światem komputerów, a Windows był tylko małym dzieckiem, programiści musieli pisać wszystko ręcznie. Internet nie był powszechnie dostępny, więc nie można się do niego zalogować, aby dowiedzieć się, jak robić fajne rzeczy w swoich grach. Tablice ogłoszeniowe (BBS) były jedynym ośrodkiem. Ktoś w małym miasteczku nie byłby w stanie sięgnąć po wiele rzeczy. Miałem szczęście, że dorastałem w stosunkowo dużym mieście i uzyskałem dostęp do wielu z tych tablic ogłoszeń. W tym momencie kompilatory nie zapewniały bardzo bogatego zestawu bibliotek, a dostęp do myszy, joysticka, a nawet klawiatury do gry był idealnie wykonywany w asemblerze. Jeśli nie znałeś wewnętrznych algorytmów i architektury komputera, byłeś naprawdę powściągliwy pod względem tego, co możesz zrobić z grą. Biblioteki istniały, ale często deweloperzy nie dostarczali kodu źródłowego, a kilka usterek w tych bibliotekach sprawiało, że były całkowicie bezużyteczne (zakładając, że w pierwszej kolejności można uzyskać do nich dostęp). Dzisiaj sytuacja zmieniła się drastycznie. Nikt nie myśli dużo o problemie z uzyskaniem danych wejściowych z myszy lub klawiatury (zakładając, że nawet odważysz się nazwać to problemem). Zamiast tego możesz skupić się bardziej na problemach związanych z grami. Często gra jest jak reklama marketingowa. Jest pełen hype i B.S. i wygląda dwa razy lepiej niż w rzeczywistości. Dla gracza zazwyczaj jest to dopuszczalne. Zazwyczaj gra wykorzystuje wiele ulepszeń, aby oszukać gracza, że jedna rzecz naprawdę jest czymś innym. Pierwszym i najbardziej oczywistym tego przykładem jest 3D. Programiści gier mogą z łatwością oszukać kogoś, że widzi grę w trzech wymiarach, ale monitor naprawdę jest 2D. Ta sama idea dotyczy takich technik, jak billboardy, które widzieliście wcześniej. Billboardy zmyślają, że obiekt ma objętość (3D), podczas gdy w rzeczywistości obiekt jest jedynie teksturą wyrównaną do ekranu. Jak zobaczysz w tym rozdziale, istnieje wiele brudnych sztuczek, których możesz użyć, aby oszukać użytkownika, który myśli, że widzi coś, co naprawdę nie istnieje. Te sztuczki są często szeroko stosowane w celu przyspieszenia procesu renderingu. Jak zobaczysz w następnym rozdziale, oświetlenie modelu w rzeczywistości nie może być obliczone w czasie rzeczywistym. W szczególności ten rozdział obejmuje:

■ Badanie potokowania 3D

■ Transformacje tekstur

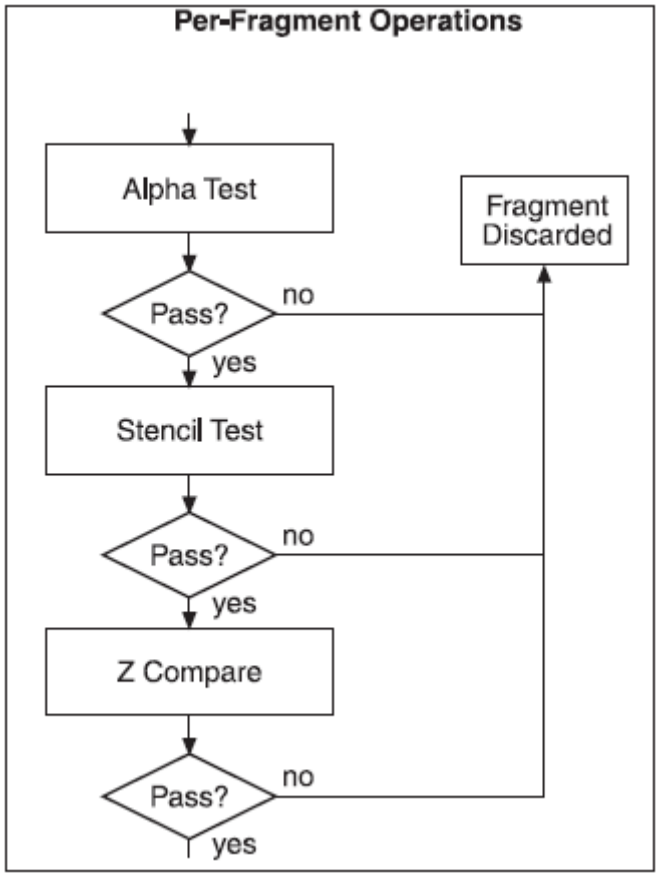
■ Sztuczki mapy sześcianu

■ Cieniowanie cienia

■ Cienie

Badanie potokowania 3D

W następnym rozdziale przyjrzymy się, jak starannie dobrać właściwy kolor w sposób globalny i perpikselowy, aby zracjonalizować kilka ładnie wyglądających scen. Ten rozdział dotyczy głównie sztuczek, które można wykorzystać na wyższym poziomie. Musisz wiedzieć, jaki sprzęt jest dostępny do renderowania. Ta sekcja jest szczególnie odpowiednia do operacji fragmentacji. Są to operacje wykonywane po wybraniu koloru piksela, ale tuż przed renderowaniem pikseli. Bufor z zerem jest taką operacją, ponieważ jest wykonywany po wybraniu koloru, ale zanim piksel zostanie faktycznie wykreślony. Rysunek 17.1 ilustruje ogólne potokowanie per-fragment.



W zależności od implementacji producenta niektóre z tych operacji mogą być dostarczane bezpłatnie (bez różnic w prędkości), a niektóre z nich mogą powodować poważne koszty. Chociaż jest to prawdą, jeśli zrozumiesz, jak się sprawy mają pod maską, będziesz w stanie wykryć, które operacje są potencjalnie bezpłatne, a które nie. Na przykład, jeśli włączysz uśpienie powierzchni i użyjesz operacji do oświetlenia, pobieranie ciał tylnych jest bezpłatne, ponieważ obliczenia oświetlenia zależą od normalnego wektora trójkąta (jak zobaczysz w następnym rozdziale). Innymi słowy, musisz wykonać obliczenia dla obu, więc istnieje szansa, że sprzedawca sprzętu zoptymalizował tę ścieżkę, aby nie obliczać normalnego wektora dwukrotnie dla tego samego wielokąta.

Test alfa

Test alfa jest dobrym rozwiązaniem, gdy chcesz obniżyć koszt rasteryzacji rurociągu renderowania. Ideą tego testu jest użycie funkcji ogólnej, takiej jak (Alpha> Constant), gdzie Alpha reprezentuje procent przezroczystości piksela. Jeśli ten test się powiedzie, wykreślasz piksel, ale jeśli się nie powiedzie, nie. Tak więc, jeśli masz obiekty, które mają prawie niewidoczną alfa (powiedzmy <5%), możesz powstrzymać się od ich renderowania, zmniejszając w ten sposób opóźnienie rasteryzacji. Zazwyczaj twój ulubiony interfejs API 3D pozwala na korzystanie z zestawu funkcji (większych, mniejszych lub równych), które pozwolą wykorzenić kilka pikseli zanim przejdą do innych operacji na fragmentach. Może to być całkiem przydatne, jeśli planujesz renderować wiele billboardów, które zazwyczaj składają się z wielu zupełnie niewidocznych pikseli; może szybko usunąć piksele, które są niewidoczne, bez wchodzenia w głąb potoku.

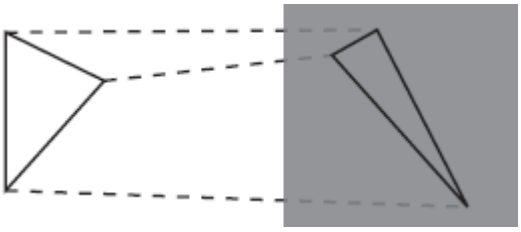
Test szablonu

Jak sama nazwa wskazuje, bufor szablonu działa podobnie do matrycy. Chodzi o to, aby bufor, który może maskować piksele, umożliwia zapisywanie pikseli tylko w regionie opisanym przez bufor

szablonu. Bufor wzorcowy jest trochę mocniejszy niż zwykły test włączania / wyłączania. Może akceptować zestaw wartości, których maksimum zależy od sprzętu. To pozwala ci renderować tylko część ekranu. Na przykład dla portalu można wypełnić piksele odnoszące się do portalu jako 1, a resztę za pomocą 0. Gdy nadejdzie czas renderowania, można ustawić funkcję, która określa, że można zapisywać tylko piksele z 1. Dlatego nawet jeśli możesz poprosić o renderowanie nowej strefy, tylko strefa widoczna w portalu będzie renderowana. Jak ilustruje ten rozdział, istnieje wiele rzeczy, które można wykonać za pomocą testu szablonu. W każdym przyzwoitym interfejsie API 3D bufor szablonu może zostać zapisany, a test przeciw niemu może zostać wykonany. Każda z tych dwóch operacji może być wyłączona lub włączona, więc możesz powstrzymać się od zapisu do bufora szablonu i tylko go sprawdzić, lub możesz po prostu napisać do niego bez względu na to, co było w nim wcześniej. Kiedy piszesz do bufor szablonu, każdy zapisany piksel może być zwiększany, zmniejszany, odwracany, wyzerowywany. . . Na przykład, jeśli chcesz wiedzieć, ile nakładek masz w swojej scenie, możesz mieć funkcję przyrostową i poprosić o renderowanie tylko pikseli, w których wartość szablonu jest większa niż 1 (to znaczy, gdzie piksel został zapisany niż raz). Podobnie można policzyć całkowitą ilość nakładania się (głębokość sceny), wykonując to rekurencyjnie, aż nic nie będzie widoczne na ekranie. Jak wspomniano w poprzednim rozdziale, nakładanie się nie jest dobrą rzeczą i powinno być minimalizowane, gdy tylko jest to możliwe. Daje to sposób na wizualizację tego nakładania się. Można również użyć bufora szablonu jako sposobu pokazania sceny przez lunetę (która jest okrągła) poprzez maskowanie bitów, które nie znajdują się wewnątrz koła. Bufor z-z zostanie tutaj pominięty. W ten sposób kończy się rurociąg fragmentów z pikselem. Jesteś teraz gotowy, aby poradzić sobie z niektórymi sztuczkami do teksturowania.

Transformacje tekstur

W poprzednich rozdziałach widziało się wiele metod, które polegały na transformacji obiekty w przestrzeni 3D. W przeważającej części transformacje te były liniowe, a zatem były reprezentowane w postaci macierzowej. Te matryce zazwyczaj transformowały wektor 4D do innego wektora 4D, i pokazano operacje takie jak obrót, translacja, skalowanie i pochylenie. Kiedy zaczynasz się interesować, kiedy zbliżasz się do renderera pikseli, można również przekształcić współrzędne tekstury. Zazwyczaj, gdy renderujesz teksturowany trójkąt, jedna współrzędna tekstury jest podany na jeden wierzchołek. Współrzędna tekstury jest następnie odwzorowywana na teksturę, a tekle (odpowiednik piksela tekstury) są odwzorowywane na piksele ekranu, jak pokazano na rysunku 17.2.



To odwzorowanie jest wykonywane na parę krawędzi, więc nie musisz się tym martwić. Ciekawą częścią jest to, kiedy możesz zmienić te współrzędne. Każdy przyzwoity interfejs 3D API jest również wyposażony w macierz transformacji tekstur, która zbyt często jest niedostatecznie wykorzystywana. To drzemie głównie dlatego, że większość programistów nie bardzo wie, jak z niego korzystać. Możesz dużo z tym zrobić, jeśli poświęcisz czas na zastanowienie się nad jego możliwościami. Macierz teksturowania przekształca współrzędne tekstury w ten sam sposób, w jaki macierz transformacji kamery / świata przekształca współrzędne 3D. Na przykład, możesz użyć macierzy tekstur, aby skalować teksturę, korzystając z transformacji, którą zobaczyłeś w części 5. Możesz też stworzyć tani efekt ruchu po niebie, jeśli z czasem przetłumaczysz teksturę i umieścisz ją pod sufitem. Teraz po prostu musisz wiedzieć, jak poprawnie go przekształcić, aby osiągnąć pożądane efekty.

Generatory współrzędnych tekstury

Większość interfejsów API 3D jest wyposażona w generatory współrzędnych tekstury. Zazwyczaj te funkcje ustawiają macierz, aby osiągnąć najczęściej używane zadania. Pozwalają zaoszczędzić sobie trudu ustawienia matrycy tekstur dla każdego trójkąta. Zamiast tego generujesz go na GPU lub na poziomie sterownika (w zależności od implementacji). Dobrze jest spojrzeć na te generatory, ponieważ nie tak wiele osób faktycznie o nich wie. Wielu programistów zwykle wprowadza je w trudny sposób lub po prostu o nich zapomina.

Generowanie Współrzędnych Tekstur Przestrzeni Obiektu

Pomysł polega na wygenerowaniu współrzędnych tekstury, które znajdują się w przestrzeni obiektu. Oznacza to, że tekstura jest nakładana na cały obiekt. Na przykład, jeśli masz czajniczek i jest on wykonany z określonego materiału, który jest podany przez teksturę. Możesz wstępnie obliczyć mnóstwo współrzędnych tekstury, aby materiał "pasował" do obiektu, lub możesz po prostu użyć tej techniki, aby zastosować teksturę do całego obiektu. To tak, jakby owijałeś teksturowany materiał wokół obiektu. Może to być przydatne w terenie. Załóżmy na przykład, że chcesz nadać konkretny kolor w zależności od jego wysokości. Możesz użyć tekstury 1D i zastosować ją do obiektu (terenu) wzdłuż osi wysokości, y. Aby uprościć problem, powiedzmy, że obiekt jest ustawiony osiowo i obrócony tak, aby był skierowany w kierunku, w którym chcesz zastosować teksturę. Jeśli musisz określić współrzędne tekstury <u, v, w> dla każdego wierzchołka, musisz tylko przypisać wartość wierzchołka do współrzędnej tekstury, zakładając, że obiekt był w dodatnim oktencie. Innymi słowy, byłoby bezpośrednie odwzorowanie od położenia wierzchołka do współrzędnej tekstury. Ponieważ większość tekstur znajduje się w 2D, można odrzucić informacje o głębi. Zatem dla wierzchołka <x, y, z> współrzędna tekstury byłaby równa <x, y>. To takie proste! Sprawa może być jednak nieco bardziej skomplikowana. Teraz musisz rozszerzyć swoje myślenie o transformacje. W poprzedniej sytuacji spojrzałeś na obiekt w przestrzeni obiektu. W większości przypadków przestrzeń obiektu zostanie wygodnie umieszczona, tak aby środek obiektu był punktem początkowym. To znacznie ułatwia obracanie wokół obiektu. Problem polega na tym, że współrzędne mają zakres od ujemnego do dodatniego we wszystkich osiach. Idealnie, tekstura powinna dokładnie pasować do obiektu (chyba że myślisz o zastosowaniu trybu powtarzania, aby powtórzyć teksturę). Zatem zakres współrzędnych tekstury powinien wynosić [0, 1], ale wierzchołki obiektu prawdopodobnie nie mają zakresu [0, 1]. Macierz transformacji tekstur powinna sobie z tym poradzić. Skaluj współrzędne tekstury obiektu, tak aby zasięg obiektu we wszystkich kierunkach wynosił 1, a środek obiektu miał wartość <0,5, 0,5>, co ostatecznie dałoby współrzędne tekstury między zakresem [0, 1]. Tak więc macierz transformacji T dla obiektu ograniczonego przez pole zasięgu <x, y, z> powinna wyglądać następująco:

$$\mathbf{T}=\begin{bmatrix} \frac{1}{x} & 0 & 0 & 0.5 \\ 0 & \frac{1}{y} & 0 & 0.5 \\ 0 & 0 & \frac{1}{z} & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ponieważ współrzędne, które podajesz, są w przestrzeni obiektu, współrzędne tekstury również powinny być podane w przestrzeni obiektu. To jest zrobione, jeśli podasz x jako współrzędną tekstury dla wierzchołka na pozycji x. Niektóre interfejsy API 3D umożliwiają określenie płaszczyzny, w której stosowana jest tekstura. W tej sytuacji chodzi tylko o określenie odległości między wierzchołkiem a płaszczyzną. W tym modelu zakłada się, że długość wektora wskazuje, jak powinna być rozciągnięta tekstura. Innymi słowy, długość normalnej płaszczyzny powinna dyktować zasięg obwiedni. Po raz kolejny gwarantuje to poprawne wykorzystanie macierzy tekstury. Pomyśl o banalnym przypadku, w którym masz trzy płaszczyzny, x = <1, 0, 0>, y = <0, 1, 0>, z = <0, 0, 1>. Odległość między punktem a płaszczyzną jest trywialnie podana przez

$$d = \left|\langle \mathbf{n}, d\rangle \bullet \langle \mathbf{p}, 1\rangle\right|$$

Tutaj <n, d> jest płaszczyzną, a p jest wierzchołkiem do oceny. Działa to tylko wtedy, gdy normalny wektor płaszczyzny jest znormalizowany. Ale w twoim przypadku, jeśli wektor normalny ma długość 2, pomnoży (skaluje) każdy komponent o 2, co jest efektem, który chcesz osiągnąć, aby związać współrzędne tekstury z zakresem [0, 1]. Wartość d jest również ważna, ponieważ obiekt powinien leżeć całkowicie po dodatniej stronie płaszczyzny. Pozwala to pozbyć się wartości absolutnej, a także całkowicie usuń tłumaczenie o 0,5. Jeśli wybierzesz d tak, że płaszczyzna <n, d> jest najbliższą płaszczyzną, dla której każdy wierzchołek znajduje się po dodatniej stronie płaszczyzny, skutecznie to osiągniesz. Ponieważ nie należy tego robić w czasie wykonywania, możesz łatwo obliczyć płaszczyznę, która zawiera dany wierzchołek p i sprawdzić, czy wszystkie wierzchołki znajdują się po jej dodatniej stronie lub na niej. Jeśli test się nie powiedzie, wybierz inny wierzchołek p i zastosuj test ponownie, aż przejdzie. Jak wspomniano w poprzednich rozdziałach, równanie, które daje wartość d to

$$d = -\mathbf{n} \bullet \mathbf{p}$$

Odległości między różnymi obliczonymi płaszczyznami można umieścić w notacji macierzowej, która ustawia matrycę teksturowania P na następującą wartość:

$$\mathbf{P} = \begin{bmatrix} a_x & a_y & a_z & -\mathbf{a} \bullet \mathbf{p} \\ b_x & b_y & b_z & -\mathbf{b} \bullet \mathbf{p} \\ c_x & c_y & c_z & -\mathbf{c} \bullet \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tutaj {a, b, c} oznaczają trzy płaszczyzny ograniczające obiekt o wyżej wspomnianych właściwościach. Wszystko to dzieje się automatycznie, jeśli używasz funkcji generowania tekstur Twojego interfejsu API 3D. Jedyne, co musisz mu dać, to zestaw samolotów kontrolujących to. Musisz jednak upewnić się, że płaszczyzny mają prawidłową długość i prawą odległość, albo wygenerowane współrzędne tekstury nie będą działać tak, jak byś tego oczekiwał. Aby owijać elementy, współrzędna tekstury u dla wierzchołka x jest określona za pomocą następującej formuły:

u = Px

Generowanie Współrzędnych Tekstur Przestrzeni Kamery

Poprzednia metoda teksturowania zawijała teksturę wokół obiektu, przyjmując współrzędne bezpośrednio z przestrzeni obiektu i manipulując nimi, aby pasowały do zakresu przyjętego przez jednostkę teksturującą. Inną przydatną techniką teksturowania jest generowanie współrzędnych

tekstury w przestrzeni kamery, zwanej także przestrzenią oczu. W związku z tym, niezależnie od tego, jak obraca się obiekt, tekstura będzie zawsze stosowana w ten sam sposób. W rzeczywistości ta technika teksturowania jest jak wstawianie zasłony przed obiektami o jednolitym kolorze. Jeśli twój obiekt był górą, którą oglądałeś z widoku z góry, możesz użyć tej techniki do teksturowania góry, przesuwając nad nią cień chmury. Tekstura musiała być tłumaczona w regularnych odstępach czasu, aby symulować wiatr w chmurach. Oczywiście działałoby to tylko wtedy, gdy patrzysz na obiekt z widoku z góry. Gdy tylko zmienisz kąt, ten sam cień zostanie wyświetlony na górze, tak jakbyś patrzył na niego z góry. Tak więc nie jest to rozwiązanie, ale jest to jeden dobry krok naprzód. Po zastosowaniu wszystkich transformacji, pożądane jest, aby wierzchołek x transformowany przez macierz transformacji świat / kamera M był powiązany z współrzędną tekstury t. Matematycznie:

$$\mathbf{t} = \mathbf{M}\mathbf{x}$$
$$\mathbf{x} = \mathbf{M}^{-1}\mathbf{t}$$

Zatem współrzędna x, z którą grałeś w sekcji współrzędnych przestrzeni przedmiotowej, powinna zostać zastąpiona równaniem podanym wcześniej. x staje się Mx przy użyciu zapisu generatora współrzędnych tekstury przestrzeni obiektowej. Oto końcowe równanie dla współrzędnych tekstury u:

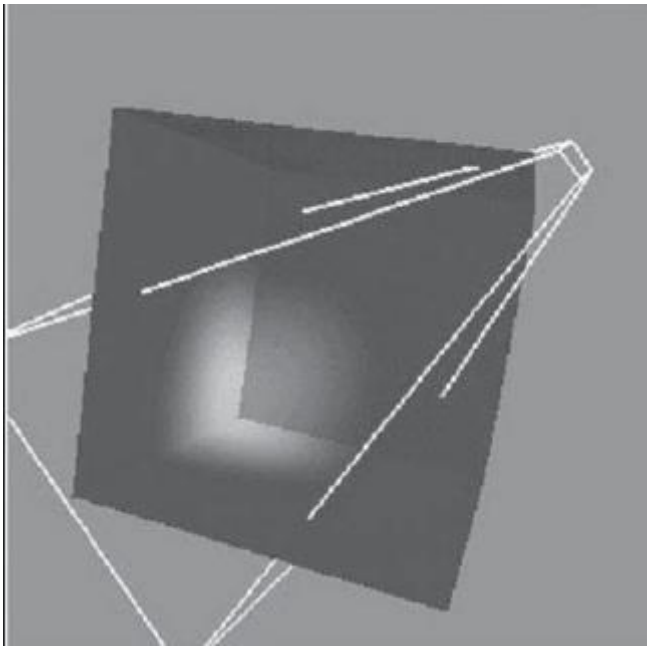
$$\mathbf{P} = \begin{bmatrix} a_x & a_y & a_z & -\mathbf{a} \bullet \mathbf{p} \\ b_x & b_y & b_z & -\mathbf{b} \bullet \mathbf{p} \\ c_x & c_y & c_z & -\mathbf{c} \bullet \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\mathbf{u} = \mathbf{P}\mathbf{M}^{-1}\mathbf{x}$$
$$= \left(\mathbf{P}\mathbf{M}^{-1}\right)\mathbf{x}$$

P tutaj jest definiowany jako kąt, pod którym ma być zastosowana tekstura. Bez względu na obrót kamery, tekstura będzie zawsze stosowana pod tym samym kątem. Jeśli chcesz, aby tekstura była stosowana w widoku z góry, tak jak w przykładzie terenu, możesz po prostu pozwolić, aby ta macierz była tożsamością? Niezupełnie, ponieważ nie pasuje do wymogu, że tekstury muszą być pomiędzy [0, 1]. Zamiast tego powinieneś użyć macierzy T, zgodnie z definicją w następnej sekcji. Więc ustawienie matrycy tekstury na PM? 1 powinno załatwić sprawę. Jeśli masz trudności z wizualizacją tego geometrycznie, pomyśl o M jako o rotacji. Załóżmy, że tekstura jest kawałkiem papieru. Weź ten kawałek papieru i obróć go. Ponieważ pracujesz w przestrzeni oczu, "tekstura" powinna nadal być tekstem nieobróconego kawałka papieru, ale sam papier powinien zarysować obróconą kartkę papieru. Aby znaleźć współrzędne tekstury, musisz usunąć obrót w przestrzeni tekstur, jak sugeruje algebra. W skrócie, jest to to samo równanie, co w przestrzeni obiektu (i nadal można wybrać kierunek dowolnie), z tym że kierunek jest statyczny względem punktu widzenia kamery. Tutaj dochodzi do odwrotności matrycy świat / kamera.

Projektywanie generowanie współrzędnych tekstury

Poprzednie dwie metody były dość podobne. Zastosowano odwróconą matrycę kamery / świata, aby wyeliminować efekt ruchów kamery, podczas gdy druga utrzymywała lokal na obiekcie. Oba można uznać za projektory ortogonalne. Innymi słowy, jeśli do tekstury używasz tylko współrzędnych 2D, co

jest najbardziej typowym przypadkiem, po prostu ignorujesz trzeci komponent. To jest cecha rzutu prostopadłego. Z drugiej strony generator rzutowych współrzędnych tekstury wykorzystuje rzut perspektywiczny. W ten sposób zniekształca scenę, tak że obiekty znajdujące się dalej od najbliższej płaszczyzny są skalowane. Pomyśl o tym jak o stożku ściętym, gdzie typowo najbliższa płaszczyzna jest mniejsza niż obszar daleki, ponieważ stożek ścięty tworzy rodzaj piramidy w z. Ideą tego typu generatora tekstur jest wyświetlanie tekstury, mniej więcej tak, jakbyś wyświetlał slajdy na ścianie. Pomysł przedstawiono na rysunku 17.3.



Piękno tej techniki polega na tym, że możesz jej użyć do rozświetlenia swojej sceny. Jeśli połączysz teksturę obiektu z fakturą światła, możesz bardzo skutecznie uzyskać dynamiczne oświetlenie sceny. Jakie to jest świetne? Oczywiście, jeśli chcesz być kulawy, możesz wyświetlać logo Batmana na niektórych chmurach. Technika ta umożliwia także realizację cieni ruchomych chmur na górze, jak wspomniano wcześniej. Tutaj nie interesuje Cię utrzymywanie tekstury względem oka, więc możesz zapomnieć o odwróconej macierzy kamery. W związku z tym prawdopodobnie powinieneś zacząć od generatora współrzędnych tekstury obiektów. Umożliwia to ustawienie statycznej tekstury na obiekcie z określonego kąta zdefiniowanego przez trzy płaszczyzny, a tym samym macierz P, jak zdefiniowano wcześniej.

$$\mathbf{P} = \begin{bmatrix} a_x & a_y & a_z & -\mathbf{a} \bullet \mathbf{p} \\ b_x & b_y & b_z & -\mathbf{b} \bullet \mathbf{p} \\ c_x & c_y & c_z & -\mathbf{c} \bullet \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{u} = \mathbf{RPx}$$

R, jak to tutaj zdefiniowano, jest matrycą projekcyjną, którą widziałeś wcześniej. Aby owijać, metoda ta jest taka sama jak metoda obiektowa, ale dodaje dodatkową projekcję, aby uwzględnić efekt głębi. Aby być dokładniejszym, poprzednia metoda wykorzystywała projekcję ortograficzną, która nie

zniekształca wartości, ponieważ zwiększa się głębokość od punktu rzutowania, podczas gdy metoda ta zniekształca wartości. Teraz możesz bez wysiłku i skutecznie osiągnąć swój cień w górach efekt.

Generowanie Współrzędnych Tekstu Omni-Sferycznych

Poprzednie tryby teksturowania generowały współrzędne o charakterze kartezjańskim. Innymi słowy, transformacja generowała tekstury o zakresie <[0, 1], [0, 1]> i była oparta na pozycji wierzchołka x. Innym podejściem do teksturowania jest wzięcie normalnego wektora tego wierzchołka. Oznacza to, że dostarczasz normalne wektory na jeden wierzchołek. Jeśli nie, po prostu nie możesz użyć tej techniki. Aby związać normalny wektor, możesz go normalizować. Tak więc, wektor normalny będzie miał promień jednego i dwóch kątów swobodny do wędrowania. Musisz również upewnić się, że zakres tekstur wynosi [0, 1], więc możesz dodać 0,5 do normalnego wektora, aby upewnić się, że wszystkie wartości są dodatnie. Jeśli weźmiesz ten sam wektor i zastosujesz na nim transformacje tekstur w przestrzeni kamery, otrzymasz coś naprawdę interesującego. Otrzymujesz funkcję, która wybiera współrzędną tekstury, w zależności od kąta wierzchołka iw konsekwencji od podstawowego trójkąta. Oto macierz T dla tej transformacji i związana z nią współrzędna dla tekstury u:

$$\mathbf{T} = \begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{u} = \mathbf{T}\mathbf{M}^{-1}\frac{\mathbf{n}}{\|\mathbf{n}\|_2}$$

Tutaj n jest normalnym wektorem wierzchołka. Macierz T, jak zdefiniowano w ostatniej sekcji, jest naprawdę banalną podstawą. Generatory współrzędnych tekstury sfer są zdefiniowane tylko dla przypadku, w którym płaszczyzny są wyrównane z kamerą. Zatem T staje się matrycą tożsamości. Ponieważ musisz przestrzegać długości normalnego wektora płaszczyzny i pozycji wyżej wspomnianej płaszczyzny, musisz skalować kierunek o połowę i dodać połowę, aby zagwarantować, że współrzędne będą miały zakres <[0, 1] , [0, 1]>. Ze względu na konstrukcję, to nigdy nie wygeneruje współrzędnych poza zakresem [1,ϕ] We współrzędnych biegunowych. Oznacza to, że twoja tekstura powinna zawsze wyglądać jak okrąg (lub, bardziej odpowiednio, dwuwymiarowa reprezentacja sfery). Tak więc w dowolnym momencie generowana tekstura jest funkcją kąta tego wektora z punktu widzenia kamery. Ten kąt jest naprawdę produktem ubocznym normalnego wektora, który daje poczucie kierunku. Normalny daje ci więcej niż kąt, ponieważ informuje również o długości. Tekstura powinna zawsze wyglądać jak okrąg lub kula, a wartość na <x, y> na fakturze jest w rzeczywistości wartością koloru dla normalnej wartości <2 [*] x - 0,5, 2 [*] y - 0,5> . To jest miłe, ponieważ oznacza, że możesz dyktować jeden piksel pod różnymi kątami. Załóżmy na przykład, że świat, na który patrzysz, jest naprawdę daleko od Ciebie, na tyle daleko, że tłumaczenie nie ma wpływu na Twój pogląd. Gwiazdy są tego dobrym przykładem. Jeśli przeszedłeś jedną milę (zakładając, że możesz to zrobić natychmiast), to nie wygląda na to, że gwiazdy w ogóle się poruszyły. W rzeczywistości gwiazdy poruszają się względem ciebie, ale ich ruch jest tak mały w porównaniu do ich odległości od ciebie, że nie dostrzegasz różnicy. Pomyśl o swoich oczach jako o kamerze. Po obliczeniu projekcji światła (w 3D) na oko (w 2D) różnica w postrzeganiu jest niewielka lub żadna. Można to zweryfikować, patrząc na formuły rzutowe z części 4, w której po raz pierwszy wprowadzono macierz rzutowania. Po uproszczeniu, dają one następujące przybliżone funkcje:

$$x' \approx \frac{x}{z}$$

$$y' \approx \frac{y}{z}$$

Ważne jest, aby zastosować projekcję, ponieważ chcesz, aby współrzędne były skalowane, gdy sięgają głębiej (jak stożek ścięty rośnie w miarę zbliżania się od bliskiej płaszczyzny do dalekiej płaszczyzny). Jak widać, jeśli gwiazdy są naprawdę daleko, z będzie bardzo duże, a stosunkowo mała zmiana w x lub y nie wpłynie zbytnio na końcową wydajność. W takich sytuacjach to, co widzisz, jest niezależne od położenia i zależy wyłącznie od kąta (lub kierunku, z którego pochodzi). Zazwyczaj nie jest to zaimplementowane w interfejsach API 3D. Zamiast tego, co zostało zaimplementowane, znane jest generowanie współrzędnych sferycznych.

Generowanie Współrzędnych Sferycznych tekstury

Szczegóły, dlaczego tak się dzieje, będą widoczne w kolejnej części, więc nie martw się zbytnio, jeśli nie wiesz, skąd pochodzą równania. Ta metoda różni się od poprzedniej tym, że wektor wejściowy, który jest podany, jest wektorem odbicia od wierzchołka. Odbicie jest jak super piłka odbijające się na ziemi bez efektu grawitacji. Kiedy super piłka uderza o ziemię, odbija się (lub zbiórek) w określonym kierunku. Ten kierunek jest tym, co ta technika przechwytuje jako współrzędna tekstury. Światło ma tę samą naturę, a zatem można użyć współrzędnych teksturowania sferycznego do symulacji błyszczących obiektów, na przykład. Błyszczące obiekty zależą tylko od dwóch rzeczy: punktu widzenia i normalnego do wierzchołka. Równanie, które będzie znacznie jaśniejsze w następnej części, jest zdefiniowane tutaj:

$$\mathbf{T} = \begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{n}' = \mathbf{M}^{-1} \frac{\mathbf{n}}{\|\mathbf{n}\|_2}$$

$$\mathbf{v} = \left(\mathbf{e} - 2\mathbf{n}'\left(\mathbf{e} \bullet \mathbf{n}'\right)\right)$$

$$m = \sqrt{v_x^{\,2} + v_y^{\,2} + \left(v_z + 1\right)^2}$$

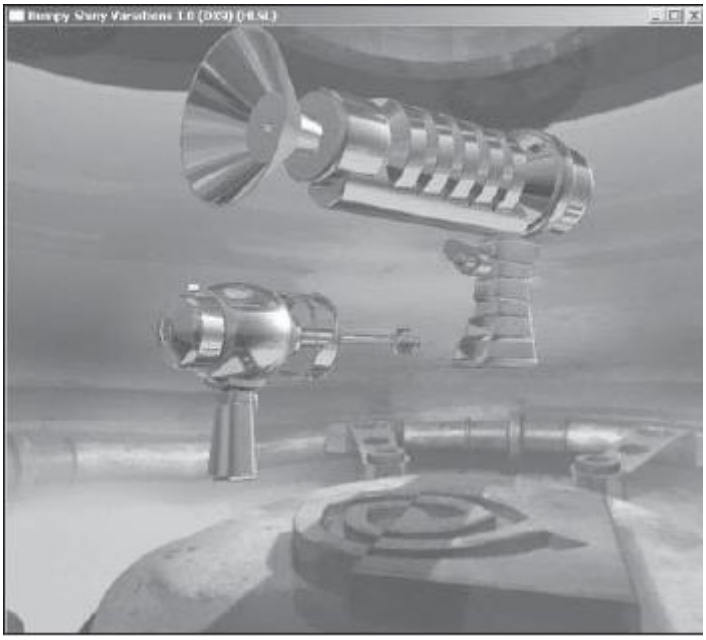
$$\mathbf{u} = \frac{\mathbf{Tv}}{2m}$$

Podobnie jak w przypadku innych metod, dotyczy to tylko tekstur 2D / 1D. Nie ma to sensu dla tekstur 3D ze względu na normalizację. Tutaj e to wektor kamery-piksela. W tym momencie możesz się zastanawiać, dlaczego musisz podzielić v przez bardzo dziwną normę. Pomaga to po prostu wcześniej wygenerować teksturę i wprowadza przyjemną kulistą relację w teksturze, przypisując wagę (to znaczy siłę ściągającą, która sprawia, że wygląda jak kula) do różnych tekseli. Jeśli spojrzysz na sferyczną teksturę mapy, wszystkie tekle zawarte w promieniu

$$r = \frac{1}{2\sqrt{2}}$$

reprezentują wektory odbicia, które próbkują światła przed kształtem, podczas gdy pozostała część promienia reprezentuje próbkę światła za kształtem. Jeśli weźmiesz samolot i wyciąć kulę na pół tak, aby płaszczyzna była wyrównana z wektorem widoku, to, co leży po dodatniej stronie płaszczyzny (w kierunku widza), znajduje się przed kulą. Krótko mówiąc, można zobaczyć odbicie tego, co znajduje się za kulą, patrząc na promień większy niż r. Może się to wydawać niemożliwe, ale następny rozdział w pełni wyjaśni to zjawisko. Zaletą tej relacji jest to, że istnieje tyle obszaru opisującego piksele za kulą, ponieważ opisuje ona przednią połowę kuli. W skrócie, każdy z nich jest reprezentowany na równi z tą konstrukcją, a m jest tym, co czyni to rzeczywistością. Aby być matematycznie poprawnym, m jest skalarem, który pozwala wyświetlać wektor normalny 3D na 2D. Wewnętrzne działania tego nie są naprawdę ważne w tym rozdziale (choć będą one w następnym). Tutaj szczególnie interesuje Cię, w jaki sposób można generować i wykorzystywać takie tekstury. Ponieważ technika ta bierze pod uwagę wektor odbicia, a nie tylko zwykły wektor, może być użyta do dowolnego obiektu, który ma charakter refleksyjny. Niemal każdy obiekt zbudowany z metalicznej powłoki, na przykład. Zaletą wektorów refleksyjnych jest to, że faktycznie zależą one od położenia obiektu. Jeśli weźmiesz łyżkę i ją przesuniesz, zobaczysz różne rzeczy na łyżce, gdy się porusza. Poprzednia technika tego nie robiła. Pozwoliło to zobaczyć to samo niezależnie od pozycji obiektu, ale obiekty odbijające nie działają w ten sposób. W ten sposób można używać tekstur sferycznych do symulacji odwzorowania środowiska. Technika ta opiera się na założeniu, że środowisko odbija się na obiekcie. Podobnie jak w przypadku poprzedniej metody, zakłada się, że środowisko jest nieskończenie odległe od ciebie dla wszystkich praktycznych zastosowań. Innymi słowy zakłada, że obiekt zawsze pozostaje stosunkowo w tej samej pozycji. (Pamiętaj o tym, że chodzisz po Ziemi i patrzysz na gwiazdy). Oczywiście, jeśli wygenerujesz sferyczną teksturę i będziesz mieć bardzo blisko siebie, przejście na drugą stronę pudełka powinno pokazać ci drugą stronę pole (we wszystkich dobrych teorii refleksyjnej). Ale z teksturami sferycznymi tekstura jest generowana tylko po jednej stronie pudełka, więc nigdy nie zobaczysz drugiej strony, nawet jeśli się tam przeniesiesz. Jest to wada tych tekstur, ale jeśli twoje otoczenie jest dość odległe od obiektu, ruch obiektu nie stanowi problemu. Innym zastosowaniem sferycznego generowania współrzędnych tekstury jest dodanie do obiektu świateł. Najważniejszym punktem omawianym tutaj jest odbicie źródła światła na obiekcie. Na przykład może to być światło z żarówki bezpośrednio odbite na plastikowej lampie. Powoduje to powstanie sferycznego światła na obiekcie, co oczywiście zależy wyłącznie od kąta i pozycji oglądania. W rezultacie możesz wygenerować mapę sferyczną z bardzo silnym punktem w środku i czystą ciemnością wokół niego. Użycie tego na obiekcie spowoduje wygenerowanie tego podświetlenia bez problemu. Generowanie map sferycznych za pomocą narzędzi do tworzenia 3D jest bardzo łatwe. Wszystko, co musisz zrobić, to stworzyć kulę w miejscu, z którego chcesz zobaczyć świat. Ze względu na budowę generowania tekstury, ta kula powinna być tak mała, jak to tylko możliwe, aby nie zamykać ani nie dotykać żadnych obiektów. Kula powinna mieć całkowity współczynnik odbicia, a tym, co chcesz uchwycić jako fakturę, jest sama kula. Możesz zachować świat wokół kuli, ale jest to bezużyteczne, ponieważ nie będzie używane do generowania współrzędnych tekstury kuli. Rysunek 17.4 pokazuje przykład mapowania środowiska zastosowanego do pistoletu;

Rysunek 17.5 pokazuje to samo z dodanymi podświetleniami, które można również osiągnąć za pomocą mapowania sferycznego.



Generowanie Współrzędnych Tekstury Sześciennej

Sferyczny generator współrzędnych doskonale nadaje się do takich rzeczy, jak światła i odległe środowiska, ale ma kilka irytujących cech. Po pierwsze, nie ma wielu próbek. Potrzebowałbyś dość dużej tekstury, aby dokładnie przedstawić środowisko. W przypadku przedmiotów takich jak metale odblaskowe i światła, nie potrzebujesz tak dużo szczegółów, więc idealnie nadaje się do tych przypadków. Inną uciążliwością jest to, że bardzo trudno ją wygenerować w czasie rzeczywistym. Jeśli przyjmiesz, że otoczenie nie jest dalekie od widoku, w idealnym świecie musisz wygenerować teksturę w każdej klatce. To działałoby do pewnego stopnia, nawet dla sferycznego texgen. Można wygenerować to, co obiekt powinien zobaczyć z danego punktu, generując teksturę, a następnie używając tej tekstury, aby renderować ją na obiekcie. Jedynym problemem jest to, że generowanie tej

tekstury nie jest czymś, co można efektywnie wykonać na sprzęcie ze względu na sferyczność charakter tekstury. Aby rozwiązać ten problem, wzdłuż przychodzi teksturowanie kostki. Zamiast używać jednej tekstury do nałożenia na obiekt sferyczny, używa się sześciu płaszczyzn definiujących sześcian. Wektor wejściowy nadal opiera się na wektorach odblaskowych wierzchołka, więc nadal można go wykorzystać do mapowania i podkreśleń środowiska. Oznacza to, że nadal pozostaje:

$$\mathbf{n}' = \mathbf{M}^{-1}\frac{\mathbf{n}}{\|\mathbf{n}\|_2}$$

$$\mathbf{v} = \left(\mathbf{e} - 2\mathbf{n}'\left(\mathbf{e} \bullet \mathbf{n}'\right)\right)$$

To, co naprawdę zmienia się, to sposób wyświetlania wektora v na zestaw tekstur. Oto sześć tekstur, więc pierwszą rzeczą, którą należy zrobić, jest wyodrębnienie tekstury, do której odnosi się Twój wektor refleksyjny. Ty decydujesz o płaszczyźnie i współrzędnej tekstury, patrząc na znak elementu o największej bezwzględnej wartości. Istnieje wiele innych sposobów na podzielenie tego; to po prostu jest jednym z nich. Po zidentyfikowaniu komponentu o największej wartości bezwzględnej wyizolowałeś problem, znajdując dwie potencjalne tekstury. Jeśli odpowiada to na przykład osi X, oznacza to, że płaszczyzna jest albo prawą, albo lewą płaszczyzną sześcianu. Podobnie, dla y, jest to górna lub dolna płaszczyzna sześcianu, a z oznacza płaszczyzny bliskie / dalekie, rzeczywisty znak tego elementu powie Ci, która z dwóch płaszczyzn jest. Na przykład, jeśli twój układ współrzędnych jest taki, że ujemny komponent x jest po lewej stronie, to mając ujemną składową x, gdy bezwzględna wartość x jest większa niż bezwzględna wartość y i z, oznacza to, że dana tekstura jest tą samą po lewej. Jeśli zastosujesz rzut perspektywiczny, jak pokazano w Części 4, możesz zachować prawo podobnego trójkąta na uwadze i możesz łatwo zobaczyć, jak każdy wektor zostanie rzucony na mapę tekstury 2D. Tak więc, jeśli masz wektor <1, 2, 3> i chcesz obliczyć współrzędne tekstury, poniżej znajdziesz sposób, w jaki to osiągniesz:

$$\mathbf{x} = \langle 1, 2, 3 \rangle$$

$$|3| > 2 > 1 \rightarrow z$$

$$\mathbf{v} = \mathbf{T}\mathbf{x}$$

$$= \begin{bmatrix} \frac{0.5}{z} & 0 & 0 & 0.5 \\ 0 & \frac{-0.5}{z} & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \frac{x}{2z} + 0.5 \\ -\frac{y}{2z} + 0.5 \end{bmatrix}$$

Projekcja w tym przypadku jest zgodna z projekcją, którą widziałaś w poprzedniej części, gdzie odległość między okiem a płaszczyzną wynosi 1 (co ma miejsce w przypadku tekstury <[0, 1], [0, 1]]>). To nie było takie trudne, prawda? Generowanie tych współrzędnych tekstury jest również bardzo proste. Ze względu na sześcienny charakter mapowania sześcianów, z fikcyjnego centrum teksturowanej kostki można wybrać bryłę, która wygląda czysto w jednej fakturze. Jest to idealne rozwiązanie do generowania map kostek w grze. Po prostu musisz ustawić poprawne obroty i wielkość stożka, a ty powinieneś być w stanie wyrenderować świat z danego widoku, aby wygenerować jedną teksturę mapy sześcianu. Jeśli zastosujesz to dla każdej z sześciu tekstur, możesz wygenerować mapę kostki w locie i użyć jej jako środowiska. Możesz spojrzeć na ten proces z prostego, kanciastego poziomu. Załóżmy, że obracasz się wokół osi Z. Są 360 stopni współrzędnych sferycznych, aby kamera mogła się obracać i powrócić do pierwotnego punktu początkowego. Z kolei mapa sześcianu ma cztery tekstury, jeśli obrócisz z. W związku z tym masz 360 stopni dla czterech tekstur, czyli 90 stopni na teksturę. Jeśli zbudujesz stożek ścięty o kącie 90 stopni lub π/ 2 rad, możesz wygenerować wszystkie cztery tekstury boczne. Następnie możesz wykonać pozostałe dwie płaszczyzny (góra / dół) i pomyślnie wygenerujesz wszystkie sześć tekstur dla mapy sześcianu. Jedynym "łapaniem" tą metodą jest konieczność renderowania świata sześć razy z sześciu różnych punktów widzenia. Oznacza to znacznie wolniejszą liczbę klatek na sekundę. Jeśli możesz stwierdzić, że niektóre tekstury nie są potrzebne, możesz uratować sobie generację kilku tekstur. Gdy tylko jest to możliwe, powinieneś używać list wyświetlania, ponieważ wszystkie one wstępnie skompilują wymagane obliczenia. Przed wywołaniem listy w tym miejscu wystarczy dodać rotację. Innym sposobem na zmniejszenie szerokości pasma i zwiększenie liczby klatek na sekundę jest zmniejszenie jakości obrazu. Na przykład możesz wyłączyć oświetlenie, użyć tańszej metody teksturowania i pomijać obiekty odległe daleko od centrum generowania mapy sześcianu. Jest to szczególnie ważne, jeśli planujesz używać tej techniki w czasie rzeczywistym z dobrą częstością klatek. W tym przypadku małe szczegóły mogą nie być tak ważne w refleksji. Możesz dodać do tego dodatkową teksturę, aby nieco rozmyć (np. Brudne okno), aby zmniejszyć efekt utraty jakości

Triki Mapowania Sześcianu

Co ciekawe, mapy sześcianu mogą być używane znacznie częściej niż na pozór. Jak już wspomniano, to, co czyni mapę sześcianu tak atrakcyjną, to to, że może przypisać do czterech wartości dla danego kąta. W poprzednim przykładzie mapy kostek zostały użyte do tekstury obiektów, a cztery przypisane wartości to czerwony, zielony, niebieski i alfa. Inną rzeczą, która sprawia, że mapy sześcianu są znacznie bardziej atrakcyjne niż mapy sferyczne, jest to, że zapewniają dobre próbkowanie kątów. Mapa sferyczna ma tylko jedną teksturę, a zatem ma słabą rozdzielczość. Mapa sześcianu oferuje sześć tekstur, a także lepszą rozdzielczość. Jak zobaczysz w poniższych podsekcjach, mapy sześcianu mogą zrobić bardzo pomocne rzeczy.

Mapa sześcianu normalizacyjnego

Używanie tablicy stałych nie jest niczym nowym w branży gier. We wczesnych dniach tablice były używane do funkcji cosinus i sinus. Nawet dzisiaj takie tabele przyniosłyby szybsze wyniki niż obliczenie funkcji za pomocą funkcji GPU, ale brak precyzji oferowanej przez te tabele sprawił, że były one nieatrakcyjne dla takich operacji. Normalizacja wektora jest dość łatwym zadaniem, a jest jeszcze łatwiejsze, gdy procesor graficzny generuje go dla ciebie w efektywny sposób. Jak pokazała ta książka, czasami dość ważne jest normalizowanie wektorów. Następny rozdział jeszcze bardziej to podkreśli. Niestety, jeśli spojrzysz na równanie normalizacji dla procesora, które ma być obliczone, nie jest ono całkiem ładne:

$$\mathbf{n}' = \frac{\mathbf{n}}{\sqrt{\mathbf{n} \bullet \mathbf{n}}}$$

Obejmuje jeden pierwiastek kwadratowy (jak brzydki), jeden podział na komponent (nieatrakcyjny) i jeden kwadrat na składnik (dopuszczalny). Nie byłoby miło, gdybyś mógł po prostu wziąć ten wektor n, rzucić go do indeksu i odzyskać znormalizowany wektor? Nie musiałbyś niczego obliczać, a po prostu musiałbyś sprawdzić. Wszystko to jest możliwe dzięki mapom sześcianu. Mapa sześcianu, jak być może pamiętasz, nie wymaga znormalizowanego wektora jako wejścia. Pobiera on arbitralny wektor n, indeksuje go do jednej z sześciu tekstur i wypluwa inny wektor, który wcześniej był używany jako składnik koloru. Innymi słowy, mapa sześcianu umożliwia mapowanie kierunku wektora do innego wektora. Całkiem potężny, nie? Jeśli wektor zawarty w mapie sześcianu jest znormalizowaną wersją wektora wejściowego, możesz uzyskać tanią funkcję normalizacji dla shaderów fragmentów. Ze względu na charakter mapy sześcianu, wartości, które są zakodowane, muszą być zapisane tak, aby pasowały do zakresu kolorów [0, 1]. W związku z tym wymagana będzie niewielka praca, aby zapewnić przestrzeganie tej zasady. Znormalizowany wektor zazwyczaj ma zakres [? 1, 1] na składnik. Jeśli więc podzielisz zakres o 2 i przetłumaczysz wartości o połowę, możesz bezpiecznie zapisać znormalizowany wektor. Następujące równanie pokazuje to:

$$\mathbf{c}_n = \frac{\mathbf{n}+1}{2}$$

Proste, prawda? Aby odzyskać wektor, musisz oczywiście postąpić odwrotnie. W takim przypadku będziesz musiał pomnożyć wynikowy wektor przez 2 i odjąć jeden. Pozwoli to uzyskać znormalizowany wektor zakodowany w mapie kostki. Ale pamiętajcie, tylko dlatego, że wektory są często używane do rzeczywistych współrzędnych lub kolorów, to nie są jedyne rzeczy, które można z nimi zrobić. Myśl nieszablonowo.

Pola Nieba

Jeśli próbujesz napisać grę na świeżym powietrzu, najprawdopodobniej natrafisz na problem renderowania odległego środowiska. Niektóre gry lubią umieszczać kilka chmur na niebie; inni lubią sprawiać wrażenie, jakbyś był blisko planety lub w górzystej scenerii. Niezależnie od tego, jaki scenariusz Cię zainspiruje, skrzynie nieba mogą przyjść z pomocą. Jeśli pamiętasz, jak generowane są mapy sześcianu, w zasadzie masz przed sobą całe rozwiązanie. Robienie migawki otaczającego świata z sześciu wzajemnie wykluczających się kątów generuje sześć tekstur dla mapy sześcianu. Ważną częścią jest to, że te sekcje wzajemnie się wykluczają. Innymi słowy, informacje o scenerii zaabsorbowane przez teksturę są unikalne. Jeśli zobaczysz piłkę w jednej z tekstur, nie powinieneś widzieć tej samej kulki w żadnej z pozostałych pięciu tekstur (chyba, że obejmuje ona dwa stożki). Co więcej, tekstury mapy kostki pobierają środowisko 3D i renderują je na kostce, niezależnie od złożoności tego świata. Co to dla ciebie oznacza? Cóż, oznacza to, że zamiast renderować niezwykle złożony świat, można wyrenderować sześcian, w którym sześć tekstur powiązanych z sześcioma ścianami kostki tworzy mapę sześcianu. Kostka pobiera próbki całego świata, dzięki czemu zobaczysz cały świat, wyświetlając tę mapę sześcianu. Ponadto, ponieważ migawki są pobierane z wzajemnie wykluczających się i sąsiednich kątów, generujesz pozornie ciągły 360-stopniowy świat z prostym polem. Aby wyrenderować ten efekt, po prostu wyrównaj sześcian, który jest wyśrodkowany wokół aparatu. Innymi słowy, rotacja jest jedyną transformacją, która wpływa na kostkę. Oczywiście należy wyłączyć buforowanie z, ponieważ można założyć, że sceneria jest nieskończenie daleko. Zaletą jest to, że w dowolnym momencie powinieneś zobaczyć co najwyżej trzy ściany sześcianu. Co więcej, w

rzeczywistości nie musisz w ogóle czyścić ekranu, ponieważ mapa sześcianu zawsze nadpisuje to, co było wcześniej. Rozmiar renderowanego sześcianu nie ma znaczenia. Ponieważ znajdujesz się w środku sześcianu, zwiększenie jego rozmiaru nie będzie miało wpływu na ostateczny wygląd. Jest to świetna sztuczka do użycia. Pole jest błyskawiczne do renderowania, a sceneria zapewnia doskonałą głębię do renderowania środowiska. Piękno tej techniki polega na tym, że czuje się ona ciągła. Na dobrej platformie renderowania nie widać nieciągłości na mapie kostki. Jeśli kojarzysz ruch obrotowy kamery z obrotowym ruchem nieba, poczujesz się tak, jakbyś naprawdę był w środowisku. Oczywiście, po raz kolejny zakłada to, że sceneria jest stosunkowo daleko od widza. Jeśli obrócisz głowę, możesz zobaczyć nieco inny kąt na obiekcie, który jest blisko. Ale spróbuj obrócić głowę, ile chcesz, i założę się, że zawsze zobaczysz ten sam kąt księżyca. To niesamowite, jak taka prosta sztuczka i odrobina matematyki mogą sprawić, że duży problem stanie się naprawdę małym problemem.

Cieniowanie Celu

Cieniowanie Celu jest tym, co służy do renderowania kreskówek. Te elementy są zaskakująco łatwe do narysowania. Są dwie rzeczy, które sprawiają, że obiekt wygląda bardzo realistycznie. Pierwszy to głębia kolorów. Im mniej masz kolorów, tym bardziej kreskówkowy jest rysunek. W kreskówce światło jest rażąco przybliżone. Innymi słowy, czasami kreskówka ma jeden kolor wypełnienia, zamiast mieć naprawdę złożony gradient koloru światła. Te ładniejsze mają 4-10 odcieni koloru, które są nadal rażąco wybierane. Innymi słowy, nie ma ładnego progresywnego gradientu od jednego koloru do drugiego. Zazwyczaj wydaje się bardziej jak łatki kolorów. Ostatnia obserwacja, która jest niezbędna do renderowania kreskówek, polega na tym, że są one w większości ograniczone czarnymi pociągnięciami. Rysunek 17.6 pokazuje przykład toon.

W tym momencie wiesz, co chcesz osiągnąć. Teraz musisz sprawdzić, jak możesz to osiągnąć. Najbardziej oczywiste jest to, że oświetlenie powinno być wyłączone. Gdy tylko użyjesz domyślnego oświetlenia w swojej scenie, nieuchronnie wprowadzisz gradient kolorów (na przykład świateł GL). Już ustaliłeś, że nie jest to coś, co chcesz dla kreskówek. Jeśli spojrzysz na sposób, w jaki animowana jest kreskówka, zawsze pojawia się bardziej dominujący kolor, a czasami ciemniejsze kolory są po prostu po to, aby wytyczać rzeczy, które wyróżniają się na danym obiekcie. Na przykład szyja może być ograniczona czarnym pociągnięciem, a sama szyja może być ciemniejsza niż twarz. Istnieją dwa rodzaje kreskówek. Niektórzy używają jednolitego koloru i czarnego obrysu do ograniczania obiektów. Najmilsi używają różnych odcieni dominującego koloru wewnątrz postaci lub przedmiotów. Spróbujmy zrozumieć pierwszy i najłatwiejszy typ. Jednym ze sposobów osiągnięcia tego jest renderowanie

obiektu w trybie siatkowym o grubym czarnym kolorze. Spowoduje to renderowanie całego obiektu za pomocą grubych linii. Po zakończeniu można renderować wypełnienie stałe do obiektu, renderując trójkąty tworzące go. Spowoduje to skuteczne renderowanie obiektu z czarną obwódką. Jedynym problemem jest to, że opuszcza granicę jako czerń. Jeśli na przykład chcesz narysować okrągły nos, nie zobaczysz czarnego obramowania wokół nosa, ponieważ został on zastąpiony wypełnieniem. Może to być dobre w niektórych przypadkach, ale w większości przypadków nie działa. Na szczęście już wiesz, jak znaleźć krawędzie, które tworzą granicę obiektu. W ostatnim rozdziale opisano technikę widoczności w celu określenia krawędzi brzegowych obiektu wypukłego. Jak się okazuje, ten sam algorytm może być również użyty dla obiektów nie wypukłych w celu znalezienia krawędzi konturu. W rzeczywistości kontur jest dowolną krawędzią, w której trójkąt z jednej strony krawędzi jest widoczny, a trójkąt po drugiej stronie krawędzi nie jest widoczny. Jeśli śledzisz ten zestaw krawędzi, możesz dokładnie określić, które z nich powinny mieć czarny kontur. Zobacz także, jak to zajmuje się nosem. Jeśli nos jest jak kula (jak to często bywa), niektóre twarze kuli będą ukryte w widoku z przodu. Oznacza to, że niektóre krawędzie konturu będą występować w tej okolicy, a tym samym mogą zostać zaczernione za pomocą tej techniki. Z powodu tej nowej techniki renderowania, będziesz musiał odwrócić kolejność renderowania. Rozumiem przez to, że najpierw należy wypełnić obiekt, a następnie narysować kontur. W tym momencie nie musisz mieć grubości większej niż 1, jak w poprzedniej sytuacji. Możesz mieć linie jako grube, jak chcesz, ponieważ porządek zmusza linię, aby była widoczna.

Jedynym haczykiem jest to, że musisz uważać na okluzję. Nie powinieneś renderować czarnych linii, które leżą pod kreskówką. W tym celu można wprowadzić przesunięcie wieloboku, które przesunie linię w kierunku kamery na tyle, że przejdzie test z-bufora. Możesz też zastosować przesunięcie samodzielnie, jeśli interfejs API 3D go nie dostarczył, po prostu przesuwając linię w kierunku kamery o małą wartość. Obejmuje to czarny kontur komiksu, ale nie pomaga w cieniu go więcej niż jednym kolorem. Kreskówka to naprawdę mniej szczegółowy przedmiot. Rozumiem przez to, że podczas gdy obiekt miałby prawdopodobnie 100 milionów kolorów, rysunek przedstawia przybliżenie brutto i może mieć tylko 10 kolorów. Innymi słowy, możesz wziąć dowolną teksturę i zmniejszyć liczbę kolorów, aby zrobić lewę. Jeśli zdecydujesz się na kreskówkę bez faktury, która ma jeden kolor, możesz chcieć wprowadzić pewną formę efektu świetlnego na kreskówce. Większość kreskówek, które biorą pod uwagę oświetlenie, również generuje jego przybliżenie poprzez dodanie ~ 4 odcieni koloru do celów oświetleniowych. Ponownie możesz odtworzyć ten efekt, oświetlając obiekt rażąco. Mapowanie sferyczne przypisuje jeden kolor do różnych wektorów odbicia. W przypadku filmów animowanych przybliżenie jest o wiele bardziej ogólne. Zamiast tego prawdopodobnie chcesz po prostu przypisać jeden kolor na kąt. Innymi słowy, fakt, że światło pochodzi z góry, dołu lub z lewej, nie jest tak ważny. Jeśli wszystko jest takie samo, możesz użyć tekstury 1D, która zależy wyłącznie od kąta między widokiem a normalnym trójkątem. W takim przypadku możesz obliczyć współrzędne współrzędnej tekstury u, używając następującego równania:

$$u = \mathbf{n} \bullet \mathbf{l}$$

Zakładając, że n i l są jednostkowe, spowoduje to odwzorowanie współrzędnych tekstury na zakres [-1, 1], ale wartości w zakresie [-1, 0] są naprawdę tylnymi i dlatego są ukryte. Daje to funkcję, która odwzorowuje kąt pomiędzy widokiem a normalnym na teksturę 1D. Aby utworzyć teksturę, musisz wybrać, który kolor powinien reprezentować kąt. Ogólnie rzecz biorąc, będziesz chciał, aby kolor opadał w sposób sferyczny, tak że tylko krawędzie, które są blisko lub które tworzą kontur, będą silniej oświetlone. Jeśli planujesz to zaimplementować w module cieniującym wierzchołków, możesz użyć normalnej funkcji oświetlenia, takiej jak w części 18, ale dodaj prostą funkcję, która konwertuje precyzyjny kolor na 3- lub 4-etapowy kolor. Cała idea cieniowania toon jest prosta. Wybierz kolor cieniowany gradientem i przekształć go w funkcję, która pozwala tylko na ograniczoną liczbę kolorów.

Cienie

Istnieje wiele sposobów dodawania cieni do gry. Niektóre rozwiązania są geometryczne, inne są oparte wyłącznie na rasteryzacji, a jeszcze inne oparte są na teksturowaniu. Moim zdaniem w tej chwili jest tylko jedna przyzwoita technika cieniowania, która jest także rdzeniem nowego silnika Doom 3. Zanim spojrzysz na to rozwiązanie, rzućmy okiem na to, co jest wciąż sprytną techniką. Nie ma sensu wdawać się w tę technikę, ponieważ ma ona drastyczną wadę, która czyni ją bezużyteczną w grze o przyzwoitej jakości.

Mapowanie cienia

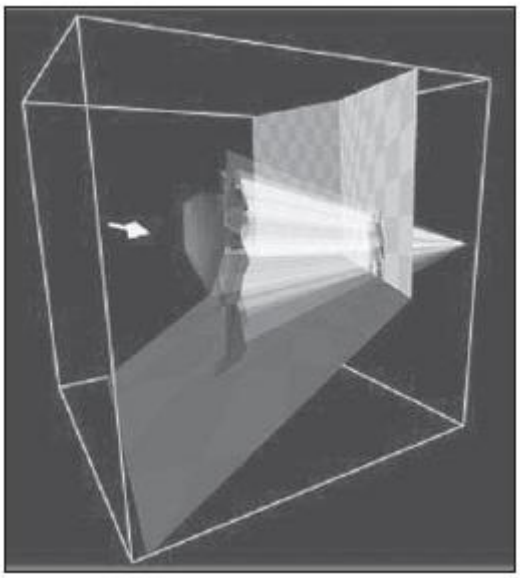
Mapowanie cieni to idea odwzorowania tekstury cienia na świecie. Możesz myśleć o nim jak o projekcji tekstury, tak jak to było zrobione wcześniej, ale z dopracowaniem. Jeśli po prostu wyświetlisz cień na świecie, nie uszanujesz porządku rzeczy. Innymi słowy, obiekt, który blokuje innych, będzie zacieniony tak samo jak obiekt, który powoduje cień. To oczywiście staje się problemem. Ta technika wymaga różnych przejść renderowania w celu uzyskania ostatecznego obrazu. Im bardziej skomplikowana jest technika, tym więcej przejść musisz zastosować. Ogólnie rzecz biorąc, chcesz zachować niską liczbę przejść, ponieważ oznacza to renderowanie sceny kilka razy. Jak widzieliśmy w generowaniu map sześcianu, może to być nawet sześć razy (siedem razy, jeśli uwzględnimy rzeczywistą scenę z punktu widzenia oka). Jeśli narysujesz scenę z punktu widzenia światła, możesz uzyskać funkcję, która mapuje współrzędne na głębokość przez zbuffer. Jeśli wyrenderujesz scenę z oka, bufor z będzie funkcją, która może odwzorować wierzchołek <x, y> na głębokość z. Tak więc, jeśli masz wierzchołek <x, y, z>, wiesz zgodnie z buforem z, że każda wartość dla z, biorąc pod uwagę x i y, jest zacieniona poza tym, co zawiera bufor z. Po renderowaniu sceny z punktu widzenia światła, jeśli weźmiesz dowolną współrzędną w przestrzeni <x, y, z>, wartość zostanie zacieniowana, jeśli wartość dla z jest większa niż wartość bufora z.

Jest to prawdą dzięki samej konstrukcji bufora z. Punkt widzenia światła jest w rzeczywistości jedynym punktem widzenia w całej scenie, w którym można zobaczyć światło i w którym cień jest całkowicie zasłonięty przez obiekt. Ten punkt widzenia czyni go idealnym do generowania mapy odniesienia głębokości (która jest z-buforem). Oczywiście renderujesz scenę z punktu widzenia oka, więc wszystko, co musisz zrobić, to znaleźć funkcję, która zamienia dowolną współrzędną w przestrzeni oczu na współrzędne w przestrzeni światła. Jeśli to zrobisz, wystarczy porównać wartość z, a będziesz w stanie stwierdzić, czy piksel w cieniu oka jest zacieniony. W tej technice nie można przejść do matematyki, ponieważ ma ona poważny problem, który czyni ją niepraktyczną lub co najmniej bardzo brzydką w grach z wszechkierunkowymi światłami. Jest dość wydajny, ponieważ sprzęt obsługuje te transformacje, a ten test można również przeprowadzić na sprzęcie. Ale jednym z dużych problemów z tą techniką jest to, że ogranicza się ona do rozdzielczości bufora z. Innymi słowy, precyzja może być tak dobra, jak precyzja oferowana w buforze z, i już ustaliłeś, że bufor Z ma słabą rozdzielczość. Może to być zadowalające w niektórych przypadkach; to naprawdę zależy od tego, jak skonfigurujesz scenę i jak szybko wszystko idzie, plus rozdzielczość. Czasami bufor z zera nie jest wystarczająco dobry, aby zagłębić się w zwykłą scenę, nie mówiąc już o mapie cieni, ale po raz kolejny zależy od tego, jak głęboka jest twoja scena. Jak widać w rozdziale dotyczącym widoczności, zbuffer może generować artefakty. Największym problemem jest to, że nie można tworzyć świateł dookolnych, ponieważ ta technika zakłada rzutowy model światła. Niestety, ograniczenia tego nie można złagodzić, chyba że chcesz renderować wiele map głębi.

Szablon Cieni

Szablony cieni rozwiązują problem z geometrycznej perspektywy. Kiedy używasz czegoś takiego jak bufor z lub tekstury jako referencje, jesteś z natury ograniczony przez rozdzielczość bufora z (24/32

bitów). Jeśli podejmiesz problem geometrycznie, potrzebujesz cięższych obliczeń, ale precyzja jest tak dobra, jak pozwalają na to pływaki. Technika cieniowania szablonu jest silnie oparta na technice okluzji obiektu. W rzeczywistości można wykazać, że każdy problem z okluzją naprawdę jest problemem cieni i na odwrót. Bardziej precyzyjnie, cień jest regionem okluzji, biorąc pod uwagę obiekt i punkt widzenia, który jest źródłem światła cieni. Co ciekawe, już przyjrzeliśmy się metodzie, która może znaleźć okluzję ograniczonych objętości ze względu na wypukły obiekt. Metoda podana w rozdziale dotyczącym okluzji najpierw znajduje granicę obiektu, a następnie generuje zestaw płaszczyzn z punktu widzenia tych krawędzi, co kończy się tworzeniem objętości okluzji. Zgodnie z podaną wcześniej definicją, ta objętość okluzji jest "zacienioną objętością" obiektu. Objętość tę zilustrowano na ryc. 17.7,



a efekt zilustrowano na ryc. 17.8.

Dostaję tutaj to, że powinieneś już znać technikę generowania cienia w cieniu obiektu, biorąc pod uwagę punkt świetlny. Technika pokazana wcześniej została zdefiniowana tylko dla obiektu wypukłego, który zamyka scenę, ale ten sam algorytm będzie działał również dla kształtu wklęsłego. Powodem, dla którego algorytm okluzji wymaga ciała wypukłego, jest to, że testowanie punktu w wypukłym ciele jest o wiele łatwiejsze. Nie można tego sprawdzić za pomocą wklęsłego korpusu, ale jest on znacznie

bardziej zaangażowany. Na szczęście w tym przypadku nie ma znaczenia, czy objętość jest wypukła czy nie. Po zidentyfikowaniu regionu cienia, zachowując zestaw płaszczyzn ograniczających cienie, problem nie został rozwiązany. Nie można renderować woluminu bez zmian w przypadku mieszania alfa po to, by przyciemnić okolicę. No, mogłeś, ale nie generowałbyś cienia tak jak ty spodziewałbym się, że tak. Innymi słowy, cień nie byłby rzutowany na kształty. Zamiast tego czułoby się, jakby przed ekranem było okno, które przyciemnia obszar. Jeśli nadal go nie widzisz, zastanów się nad prostym scenariuszem, w którym jest światło z obiektem blokującym. Załóżmy, że ziemia jest płaska. Co powinieneś zobaczyć, to cień w ziemi, a nie rzeczywisty stożek cienia. W związku z tym potrzebny jest sposób maskowania pikseli, które nie są zacienione. Jak sugeruje nazwa techniki, bufor wzorcowy jest tutaj twoim wybawcą. Technika polega na prostej obserwacji. Przy wypukłym cieniu w kształcie cienia, jeśli obiekt jest zacieniony, dzieje się tak, ponieważ przynajmniej jedna powierzchnia obiektu znajduje się za wielokątami, które są widoczne z punktu widzenia oka, a przed wielokątami. które są cofnięte z punktu widzenia oka. Jest to po prostu inny sposób powiedzenia, że obiekt jest "wewnątrz" cienia. Algorytm najpierw wymaga renderowania rzeczywistej sceny tak jak jest z włączonym buforem głębokości. Sztuczka polega na znalezieniu każdego wierzchołka, dla którego przednia powierzchnia cienia ze stożkiem ściętym jest przednim wielokątem. To da ci zestaw wierzchołków, które są zawarte w cieniu frustum i które są widoczne. Nie ma sensu ocieniać obszaru, jeśli przed nim znajduje się obiekt, który zamyka całą scenę. Biorąc pod uwagę ten zestaw, usuń wszystkie wierzchołki, dla których widoczna jest tylna powierzchnia cienia ściętego, a ten zestaw wierzchołków będzie tymi, które powinny być zacienione. Może to zabrzmieć trochę myląco, ale to naprawdę proste. Wspomniano wcześniej, że obiekt, a dokładniej piksel, powinien być zacieniony, jeśli leży za przednimi powierzchniami stożka ściętego i przed tylnymi częściami stożka ściętego. Innymi słowy, piksel znajduje się w cieniu cienia. Jeśli weźmiesz wszystkie wierzchołki, które znajdują się za cieniami cienia i usuniesz wierzchołki, dla których nie ma piksela przed płaszczyznami cienia, pozostajesz z pikselami, które znajdują się w cieniu cienia, a więc muszą być zacienione. Jest jeszcze jeden mały problem, z którym musisz sobie poradzić. Wszystko to dotyczy tylko wypukłego cienia. Co się stanie, jeśli cień nie jest wypukły? Pomysł nadal działa, ponieważ wklęsły wielokąt widziany z boku może być postrzegany jako zestaw wypukłych wielokątów. Innymi słowy, jeśli weźmiesz linię, która biegnie od kamery do nieskończoności, linia może wejść w cień cienia, ale musi ona z niego wyjść przed potencjalnym wejściem do innej części cienia. Ten sam efekt można osiągnąć, śledząc, ile razy ta wirtualna linia przekracza granice. Załóżmy, że zaczynasz od renderowania przednich ścian cienia. W przypadku piksela <x, y> na ekranie za każdym razem, gdy wielokąt cienia znajduje się powyżej wartości w zbufferze (bliżej kamery), należy zwiększyć bufor szablonu o jeden dla <x, y>, sugerując, że raz wszedłeś w sylwetkę od frontu. Ponieważ w rzeczywistości nie chcesz, aby cień w kształcie cienia był renderowany tak jak jest, powinieneś renderować stożek ścięty, tak jak to tylko modyfikuje bufor szablonu, a nie bufor koloru lub głębi. Przy drugim przejściu będziesz musiał wyrenderować cienie w cieniu, w takim przypadku będziesz musiał dopasować przednie twarze do płaszczyzn. W związku z tym należy tak ustawić reguły, aby bufor szablonu był mniejszy, jeśli cień cielesny jest widoczny. Oznaczałoby to, że linia ta przekroczyła obie granice stożka ściętego, a piksel nie jest zacieniony. Jeśli obiekt znajdowałby się między przednią a tylną płaszczyzną cienia, to tylne części ściętego stożka nie zmniejszyłyby objętości cieni, ponieważ głębokość przedmiotu znajdowała się powyżej głębokości ściętego stożka, a zatem pozostawiłaby 1 w ściętym stożku. Gdy skończysz, masz bufor szablonu z wartościami zerowymi, gdzie piksele nie powinny być zacienione, oraz wartościami niezerowymi, gdy piksele powinny być zacienione. W tym momencie możesz renderować zestaw cieni przy użyciu tego samego bufora szablonu, aby zaoszczędzić trochę czasu renderowania. Gdy już w pełni rozpoznasz, które piksele są zacienione za pomocą bufora szablonu, wszystko, co musisz zrobić, to zastosować stały mieszany alfabet wielokąta (ciemny przezroczysty wielokąt), aby uzyskać efekt cieniowania. To wszystko. Brzmi to prawie zbyt łatwo i tak jest. Jeśli kamera znajduje się w cieniu cienia, masz problem.

Szablon zmieni się tylko raz, ponieważ niektóre z przednich ścian mogą być ukryte za kamerą. Jest to oczywiście problem, ale można go łatwo naprawić. Zamiast sprawdzać, czy wielokąty cieniotki przechodzą test z-bufora, sprawdźmy przeciwnie. Innymi słowy, sprawdźmy, czy test się nie powiedzie. Jeśli test nie powiedzie się dla powierzchni czołowych, możesz zwiększyć wartość, a jeśli test nie powiedzie się dla warstw, możesz zmniejszyć. Co jeśli test nigdy nie zawodzi (linia wchodzi i wychodzi ze stożka ściętego, a bufor szablonu nie jest w rzeczywistości wzruszony)? Jest to tylko odwrotność poprzedniego testu, który oczywiście nadal będzie działał, więc w jaki sposób ci to pomaga? Przypuśćmy, że cień ścięty jest kształtem zamkniętym. Znajdowanie się w środku tego stożka spowoduje przełączenie fragmentu wielokąta cieni szablonu, ale ponieważ przednia powierzchnia nie jest widoczna, nie będzie go kręcić. Test sprawdza się więc doskonale zarówno wewnątrz, jak i na zewnątrz aparatu. Teraz jedynym problemem, który pozostaje, jest ustalenie, w jaki sposób wiązać cień cienia. To nie jest trudne. Pierwszym krokiem jest zamknięcie bryły ściętej z trójkątów obiektu zamykającego. Możesz po prostu użyć trójkątów frontface obiektu, aby ograniczyć objętość. Ta czapka jest zbliżona do bliskiej płaszczyzny, ponieważ jest to najbliższa płaszczyzna z punktu widzenia światła. Teraz musisz znaleźć to, co jest zbliżone do odległego samolotu. Jest to dość skomplikowana sprawa, ponieważ można natknąć się na bardzo nieprzyjemne sprawy. Na przykład, jeśli światło dookólne znajduje się w doniczce z otworami, samolot nie wykona zadania, ponieważ dwie dziury mogą być naprzeciwko siebie. Zamiast tego można projektować odwrotności obiektu okluzji w nieskończoność. Ale jak dokładnie możesz projektować w nieskończoność? Coś nie wygląda dobrze! W sekcji rzutowania tekstury pominięto jedną rzecz. Typowa matryca projekcyjna w każdym przypadku nie działa zbyt dobrze. Dzieje się tak dlatego, że matryca rzutowania, którą widziałeś, faktycznie spina wszystko do płaszczyzn ściętego stożka, a więc także klipów do płaszczyzny Z. Oczywiście, jeśli projektujesz teksturę na ścianie, tak naprawdę nie ma żadnych ograniczeń co do długości projekcji, co sprawia, że typowa matryca projekcji jest niepożądana. Oto typowa matryca rzutowania perspektywicznego P:

$$\mathbf{P} = \begin{bmatrix} \frac{2e}{b-a} & 0 & 0 & 0 \\ 0 & \frac{2e}{d-c} & 0 & 0 \\ \frac{b+a}{b-a} & \frac{d+c}{d-c} & \frac{-f-e}{f-e} & -1 \\ 0 & 0 & \frac{-2ef}{f-e} & 0 \end{bmatrix}$$

Tutaj d jest odległością od oka do bliskiej płaszczyzny, a f jest odległością do dalekiej płaszczyzny. Jeśli oceniasz, co się dzieje, gdy dalekie płaszczyzny zmierza w kierunku nieskończoności, możesz skutecznie osiągnąć to, czego szukasz:

$$\lim_{f\to\infty}\mathbf{P}=\begin{bmatrix}\frac{2e}{b-a} & 0 & 0 & 0\\ 0 & \frac{2e}{d-c} & 0 & 0\\ \frac{b+a}{b-a} & \frac{d+c}{d-c} & -1 & -1\\ 0 & 0 & -2e & 0\end{bmatrix}$$

Część 7, nie przyjrzała się dokładnie ocenie limitu z intuicyjnej perspektywy. Intuicyjna metoda jest zawsze dobra do stawiania przyzwoitych domysłów, ale nie gwarantuje, że jest to rozwiązanie. Musiałbyś przejść przez rygorystyczną matematykę, jeśli chcesz ustalić z całą pewnością, że twoje przypuszczenia były poprawne. W tej sytuacji można zauważyć, że skoro f dąży do bardzo dużych liczb, praktycznie nie ma znaczenia. Jeśli zastąpisz e wartością 0, aby reprezentować jej nieistotność, otrzymasz stosunek -1. Jeśli zastosujesz tę samą logikę w drugim przypadku, będziesz miał błędne domysły, a więc znaczenie weryfikacji twojego domysłu. W tym konkretnym przypadku możesz podzielić górną i dolną część ułamka za pomocą f. Da ci to:

$$\begin{aligned}\lim_{f\to\infty}\frac{-2ef}{f-e} &= \lim_{f\to\infty}\frac{-2e}{1-\frac{e}{f}}\\ &= \frac{-2e}{1-0}\\ &= -2e\end{aligned}$$

W rzeczywistości metoda ta może być również używana w poprzednim przypadku, a zobaczysz po raz kolejny, że wynikiem jest -1. Ze względu na precyzję, niestety ta matryca może dać znormalizowane współrzędne z komponentem Z nieco większym niż 1. Dlaczego powinieneś się tym przejmować? Cóż, sprzęt przeprowadza transformację z przestrzeni oczu na klip-przestrzeń w celu przycięcia punktów. Normalizuje to wektor, a jeśli twoja nieskończona projekcja ma stosunkowo słabą precyzję, może zostać wykryta jako obcięta, jeśli nie chcesz, aby była w ogóle zaciśnięta w całym z. Aby to naprawić, możesz dodać małą stałą dodatnią do zakresu. Dlatego zamiast mieć zakres [-1, 1] dla z, możesz mieć zakres, który jest lekko przesunięty [-1, 1 + ε]. Jeśli wykonujesz dla niego matematykę, powinieneś uzyskać ostateczną macierz projekcji:

$$\lim_{f\to\infty}\mathbf{P}=\begin{bmatrix}\frac{2e}{b-a} & 0 & 0 & 0\\ 0 & \frac{2e}{d-c} & 0 & 0\\ \frac{b+a}{b-a} & \frac{d+c}{d-c} & \varepsilon-1 & -1\\ 0 & 0 & n(\varepsilon-2) & 0\end{bmatrix}$$

W związku z tym, jeśli chcesz zaoszczędzić sobie trudu zbudowania macierzy od zera, możesz załadować typową macierz projekcyjną i zmodyfikować dwa parametry, które różnią się od typowej macierzy rzutowania perspektywicznego. Mianowicie są to te, w tym epsilon. Alternatywnie, niektóre karty faktycznie obsługują tryb zwany zaciśnięciem głębokości. Umożliwia to zaciśnięcie znormalizowanych wektorów w zakresie [-1, 1]. W ten sposób nie musisz martwić się o epsilon, ale musisz zmodyfikować matrycę, zanim zostanie zmodyfikowana przez epsilon. Innym podejściem opartym na tym samym pomyśle jest renderowanie sceny za pomocą różnych modeli oświetlenia, o których mowa w części 18. Pomysł jest prawie taki sam, ale zamiast używać wielokąta do przyciemnienia / ocienienia obszaru, można renderować scenę raz z oświetlenie otoczenia - czyli światło, które jest wspólne dla zacienionych i nie cieniowanych regionów. Następnie należy wyrenderować objętość cienia, aby określić, który obszar jest zacieniony przy użyciu bufora szablonu, ale bez modyfikowania koloru na ekranie. Na koniec możesz ponownie renderować scenę, włączając wszystkie światła. Dzięki buforowi szablonu, powinno to tylko renderować obiekty, które nie znajdują się w cieniu, a zatem powinny je prawidłowo oświetlać. Niefortunną rzeczą w tej technice jest to, że wymaga ona renderowania wszystkiego dwa razy (lub trochę mniej przy pewnych optymalizacjach) i wymaga znacznie więcej pracy niż po prostu przyciemnienie obszaru. Dodatkowym efektem jest to, że scena jest oświetlona poprawnie, jeśli chodzi o fizyczny opis światła.