XV. Wyznaczanie widoczności: Świat niewidzialny

Albert Einstein powiedział kiedyś: "Każdy inteligentny głupiec może sprawić, że rzeczy będą większe, bardziej złożone i bardziej gwałtowne. Potrzeba odrobiny geniuszu - i dużo odwagi - aby ruszyć w przeciwnym kierunku. "Nie potrafię wymyślić lepszej filozofii podczas projektowania algorytmów gry. Każdy głupiec może zbudować silnik 3D, ale stworzenie dobrego, wydajnego silnika 3D wymaga geniuszu. Pomysł ten dobrze pasuje również do wykrywania kolizji. Możesz leniwym sposobem, realizując wszystko jako kule i zapominając o koncepcji czasu, lub możesz wybrać bardziej dokładny model, który wymaga czasu. Nie chcę, żebyś odniosła wrażenie, że poprzednia metoda nie jest dobra. Wręcz przeciwnie, może być bardzo dobre dla niektórych rzeczy. Ale może przynieść katastrofalne wyniki, na przykład jeśli chcesz przedstawić ciężarówkę jako kulę. Kolizja wypadłaby całkiem sporo na szczycie ciężarówki. Obwiednia byłaby o wiele lepszym wyborem. Widoczność to problem, który również zasługuje na uwagę. Możesz wybrać leniwy sposób robienia rzeczy, czyli renderowania każdego obiektu na świecie, lub możesz wybrać bardziej inteligentny sposób, który polega na usunięciu fragmentów świata, które nie są widoczne dla użytkownika. Taki jest cały pomysł za tą częścią. Kiedy patrzysz na potok 3D, masz trzy duże segmenty, które mogą spowolnić twoją grę. Pierwszym z nich jest procesor, w którym wykonywane jest przetwarzanie. Drugim jest przetwarzanie geometrii GPU. Ta jednostka odpowiada za wszystkie transformacje. Trzeci i ostatni to rasteryzer, który zajmuje się rzeczywistym renderowaniem trójkątów na ekranie. W tym rozdziale omówiono optymalizację drugiego poziomu poprzez zmniejszenie geometrii przetwarzanej przez procesor graficzny. Istnieją szerokie klasy podejść do tego problemu:

■ z bufor

■ Odrzucanie tylnych ścian

■ Usuwanie brył

■ Wykrywanie projekcji (objętość graniczna)

■ Usuwanie obiektów

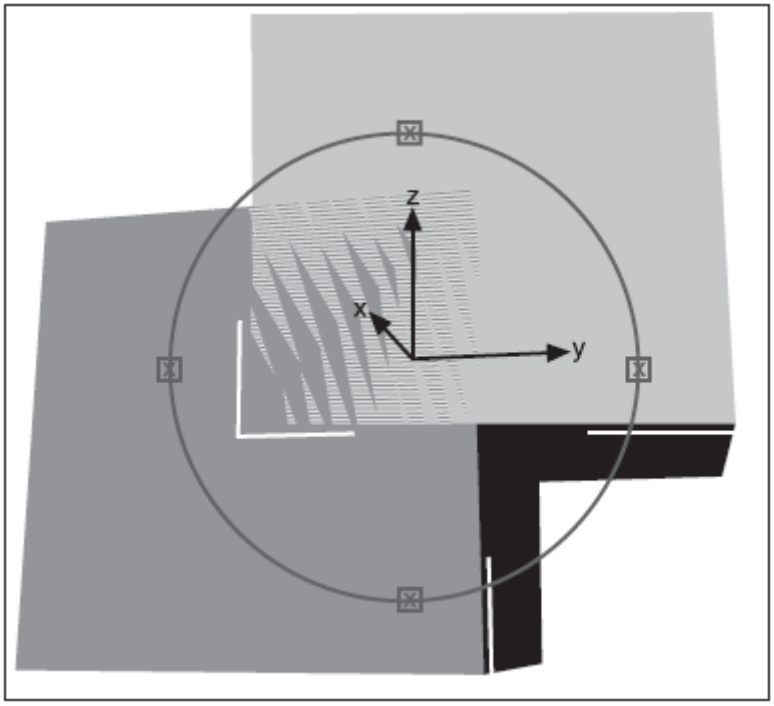
Określanie widoczności na poziomie GPU

Najpierw wszystko. Powinieneś zacząć od zrozumienia, jak działa GPU, zanim spojrzysz na optymalizacje, które możesz wykonać wcześniej. Typowy procesor graficzny ma dwie warstwy widoczności, które wykorzystuje do wykrywania widoczności prymitywów: bufor z-z i uśpienie z tyłu.

Z-Bufor

Być może już wiesz o z-buforze , ale prawdopodobnie są rzeczy, o których nie wiesz. Bufor Z to bufor pamięci w GPU, który zawiera 16/32 bitów danych w przeliczeniu na piksel. Jest to najniższy poziom odrzucenia, jaki można zrobić, ponieważ znajduje się na poziomie pikseli. Bufor z-z utrzymuje śledzenie wartości z dla każdego piksela i przechowuje tę wartość w buforze odpowiadającym temu konkretnemu pikselowi. Gdy GPU renderuje elementy, sprawdza, czy piksel, który ma zostać narysowany, znajduje się przed lub za ostatnio zapisanym pikselem. Jeśli jest z przodu, piksel jest renderowany, usuwając poprzedni kolor. W dzisiejszych interfejsach API 3D zazwyczaj można kontrolować regułę zapisu ustawioną za pomocą bufora z. Możesz poprosić go o renderowanie pikseli tylko wtedy, gdy składnik z jest mniejszy lub większy. Z punktu widzenia widoczności poprzedni przypadek jest najbardziej atrakcyjną opcją. Tego rodzaju test jest dość kosztowny w oprogramowaniu, ponieważ każdy pojedynczy piksel musi zostać przetestowany, aby można go było wydrukować. Oczywiście można go wyłączyć, ale nie ma sposobu, aby ustalić kolejność, która działa w każdym

możliwym przypadku. Nawet stary algorytm malarski, polegający na malowaniu obiektów od tyłu do przodu, tak, że rzeczy najdalsze są oddawane jako pierwsze, cierpi, gdy dwa wielokąty przecinają się i części obu są widoczne. Z drugiej strony, w implementacjach sprzętowych, bufora Z przychodzi za niewielką cenę. Czasami może to za darmo, w zależności od dostawcy ardware i jego realizacji, co czyni go opłacalną opcją, ale ogólnie rzecz biorąc, nie jest to najbardziej optymalna forma określenia widoczności, ponieważ wymaga wysłania każdego obiektu w dół karty, nawet jeśli niektóre obiekty nie muszą być przekształcane. Niestety, z-buffer ma problemy. Pierwszy problem polega na tym, że jest on ograniczony precyzją punktów zmiennoprzecinkowych. Na przykład, jeśli weźmiesz dwa trójkąty i narysujesz jeden na drugim, matematycznie trójkąty znajdują się w tym samym miejscu w dowolnym punkcie wzdłuż obydwu. Ale ze względu na precyzję wykazywaną przez bufor Z, czasami pojawiają się części pierwszego trójkąta, podczas gdy części drugiego trójkąta będą się pojawiać. Zilustrowano to na rysunku 15.1.



Jest jednak jedna rzecz, którą powinieneś wiedzieć o buforze z. Nie zachowuje się liniowo. Obiekty znajdujące się bliżej aparatu nie ucierpią z artefaktów pokazanych na Rysunku 15.1, tak jak obiekty znajdujące się dalej od aparatu. Sam w sobie nie jest to złe, ale oznacza, że twoje obiekty nie będą spójne. Ale dlaczego tak się dzieje? To naprawdę proste. Jeśli przypomnisz sobie, jak działa matryca projekcji, musisz podzielić składniki przez z na projekt z 3D na 2D. Wartość bufora z jest obliczana w następujący sposób:

$$ZBuffer(z) = \frac{2^n (ZFar \cdot ZNear - ZFar)}{z(ZNear - ZFar)}$$

Tutaj n jest liczbą bitów dla bufora z, a Zfar i Znear są dwiema płaszczyznami <0,0, ±1, Z [Near / Far]>. Jak widać, jeśli z wzrasta o 1, wartość przechowywana w buforze z oczywiście nie wzrośnie o 1, ponieważ relacja między Zbufferem i z jest taka, że jest proporcjonalna do 1 / z. Funkcja 1 / z zapewnia większą dokładność niższym wartościom niż w przypadku większych wartości (czyli najbliżej i najdalej). Z drugiej strony, niektóre karty zapewniają bufor w, który przechowuje wartości w trybie 1 / z zamiast

z, jak pokazuje to równanie. Tak więc, użyjesz równania dla bufora z, gdzie z jest proporcjonalne do bufora z, a więc mapowania liniowego. Z buforem w, dokładność jest liniowa. Co to wszystko oznacza dla twojej gry? Zasadniczo oznacza to, że istnieje duża dokładność tam, gdzie nie jest to naprawdę potrzebne. Fajnie jest mieć większą dokładność elementów, które są ci bliższe, ale w przypadku gier, które są głównie plenerowe, większość obiektów znajduje się stosunkowo daleko. W tym momencie przydatne jest posiadanie czegoś, co zapewnia większą precyzję na większe odległości. Możesz tylko przesunąć najbliższy samolot o ścięty kawałek dalej. Jeśli odepniesz tę płaszczyznę ściętego stożka, zmniejszysz wartości bufora z, aby obiekty znajdujące się dalej miały trochę większą dokładność. Oczywiście, powinieneś to zrobić tylko do rozsądnego punktu, inaczej będziesz brakować części sceny. Kluczem jest tu eksperymentowanie. To naprawdę jest konfiguracja na grę.

Odrzucanie tylnych ścian

Odrzucanie tylnych ścian jest jedną z najprostszych i najprostszych form odrzucenia do wdrożenia. Chodzi o to, że większość obiektów jest zamknięta, więc za jednym razem widać tylko jedną stronę powierzchni obiektu. Weź kulę, na przykład. Nigdy nie widzisz wnętrza kuli, dopóki twój system wykrywania kolizji nie zniknie. Ale jak to się robi? Głównie opiera się na właściwościach określonych przez produkt krzyżowy. Przypomnij sobie, że produkt krzyżowy jest bardzo wrażliwy na zamówienie, które mu dajesz. Jeśli obliczysz iloczyn dwóch wektorów, a następnie obliczymy iloczyn krzyżowy tych dwóch wektorów, odwracając ich kolejność, otrzymamy ten sam wektor, ale w przeciwnym kierunku. Zależy to od implementacji twojego dostawcy sprzętu, ale test w GPU może być wykonywany o jeden poziom powyżej bufora z. Bufor z przeprowadzono na etapie rasteryzatora i ten test przeprowadzono na etapie geometrii. Dokładniej, jest to wykonywane zaraz po przekształceniu trójkąta w trójkąt 2D, w którym zajmujesz się tylko wierzchołkami <x, y>. Niektóre inteligentniejsze urządzenia mogą wykonywać operacje usuwania ciał przed przekształceniem obiektu na 2D, co omówiono w dalszej części tej części. Jeśli stwierdzimy, że każdy widoczny trójkąt będzie przechowywany zgodnie z ruchem wskazówek zegara, obliczenie składowej z produktu krzyżowego $<x_1, y1, 0>$, $<x_2, y2, 0>$ powie, czy wielokąt jest zgodny z ruchem wskazówek zegara, czy w lewo patrząc na znak elementu. Co ciekawe, gdy wielokąt nie jest skierowany w stronę kamery, jego projekcja generuje trójkąt przeciwny do ruchu wskazówek zegara. Ma to sens, jeśli użyjemy następującego trywialnego przykładu: weź trójkąt, który jest widoczny, i obróć go o 180 stopni wokół dowolnej osi. Gdy to zrobisz, trójkąt pokazuje swoją twarz. Jeśli spojrzysz również na kolejność wierzchołków, zauważysz, że kierunek zmienił się z ruchu wskazówek zegara na przeciwny do ruchu wskazówek zegara. Ponownie, jest to łatwe do wykrycia z krzyżowym produktem, patrząc na znak.

Przycinanie Brył Ściętych

Inną formą przycinania, którą wykonuje sprzęt, jest przycięcie brył. Wprowadza to inną przestrzeń, zwaną przestrzenią przycinania. Widok mapy perspektywicznej przekształca pole w piramidę, jak omówiono w części 5. W konsekwencji, po rzucie, punkt powinien mieć zakres <[?- , 1], [-1, 1], [? 1 , 1]>. W tym momencie sprzęt może spiąć trójkąt, jeśli trójkąt znajduje się całkowicie poza punktem widzenia. Ten test jest bardzo łatwy do wykonania, ponieważ jest prostym dodatkiem do potoku 3D, który już jest obecny. Ponadto test ma bardzo niski koszt, ponieważ wystarczy go przetestować na pudełku ustawionym w osi. Jedynym problemem jest to, że jest to wykonywane tylko na zasadzie trójkąta, co oznacza, że nadal pracujesz na dość mikroskopijnym poziomie.

Obciążenie CPU

Test odrzucenia tylnej ściany jest wyraźnie lepszy niż bufor Z, ponieważ pozwala pominąć rastrowanie dla niektórych wielokątów. Podobnie, test uboju ścięgna jest lepszy niż oba poprzednie, ponieważ klipy są na wyższym poziomie. Ale czy to naprawdę wystarczy? Czasami obiekty są całkowicie niewidoczne

dla kamery, w takim przypadku przekształcasz cały obiekt tylko po to, aby zauważyć na raczej niskim poziomie, że nie jest on widoczny. Jedynym rozwiązaniem jest cofnięcie się o jeden krok i sprawdzenie, co można zrobić na etapie procesora, aby zmniejszyć liczbę trójkątów wysyłanych do etapu transformacji. Im szybciej możesz dowiedzieć się, czy obiekt jest widoczny, czy nie, tym lepiej. Jeśli musisz przejść całą drogę do bufora z, aby to zrozumieć, potrzeba znacznie więcej pracy i marnowania przepustowości GPU. Jedyne, co musisz zachować ostrożność, to to, że procesor nie staje się wąskim gardłem. Jak zawsze, musisz zachować ostrożność, aby zminimalizować opóźnienie. Jeśli procesor staje się wąskim gardłem, równie dobrze możesz wysłać więcej trójkątów do procesora graficznego i zwolnić procesor. Zasadniczo jednak tak nie jest. Przyjrzyjmy się różnym sposobom, w jaki można uzyskać ubożenie.

Usuwanie bryły ściętej

Bez wątpienia najbardziej oczywistym i prostym testem, jaki możesz wykonać, jest sprawdzenie, czy przedmioty znajdują się w ciele. Przypomnij sobie, że stożek ścięty może być postrzegany jako zestaw sześciu płaszczyzn, które opisują widoczny obszar z punktu widzenia kamery na świecie. W związku z tym, jeśli chcesz sprawdzić, czy obiekt jest widoczny z punktu widzenia punktu ściętego, wystarczy, że uzyskasz współrzędne światowe obiektu i przetestujesz współrzędne świateł, jak pokazano w części 5. O co w tym fajnie? Określenie widoczności polega na tym, że można zawyżać widoczny obszar bez zmiany wyniku. Innymi słowy, możesz twierdzić, że obiekt jest widoczny, nawet jeśli tak naprawdę nie jest, a wynik nie ulegnie zmianie. Nie oznacza to, że powinieneś wykonać naprawdę gruboziarnisty test, ponieważ jest to sprzeczne z ideą optymalizacji rurociągu. Idealnie, szukasz testu, który można wykonać skutecznie i który wykrywa większość niewidocznych obiektów. Jest to przeciwieństwo wykrywania kolizji, gdzie liczy się precyzja. Zmiana dokładności wykrywania kolizji może zmienić wynik, ponieważ nic innego nie zajmie się kolizją, jeśli nie wykryjesz istniejącej.

Test Płaszczyzna – Bryła Ścięta

Wykrywanie przecięcia płaszczyzny ze stożkiem ściętym jest łatwe, jeśli używasz pół-płaszczyzn. Innymi słowy, możesz sprawdzić, czy wszystkie rogi ściętego stożka znajdują się po tej samej stronie płaszczyzny. Jest to dobre rozwiązanie, ale nie jest bardzo chude, ponieważ sugeruje, że osiem wierzchołków musi być sprawdzonych względem płaszczyzny. Zamiast tego znacznie lepiej przekształcić przestrzeń w przestrzeń przycinającą. Tak się złożyło, że macierz projekcji została zdefiniowana jako macierz, która przekształca zwykłe jednolite pole (w przestrzeń przycinającą) w coś, co znacie jako trójwymiarową bryłę (w przestrzeni ekranu). Odwrotność tej macierzy działa odwrotnie. Zajmuje cokolwiek w przestrzeni ekranu (lub rzutowanej przestrzeni) i przekształca przestrzeń w taki sposób, że stożek ścięty staje się polem, a macierz transformacji odpowiednio zmienia wszystkie obiekty względem stożka ściętego. Jeśli przypominasz sobie, w jaki sposób transformacje są planowane, nie powinieneś mieć problemu z zaakceptowaniem, że formuła do konwersji płaszczyzny <n, D> z rzutowanego na obszar rzutowania jest

$$\langle \mathbf{n}, D \rangle' = \left((\mathbf{PC})^{-1} \right)^{\mathrm{T}} \langle \mathbf{n}, D \rangle$$

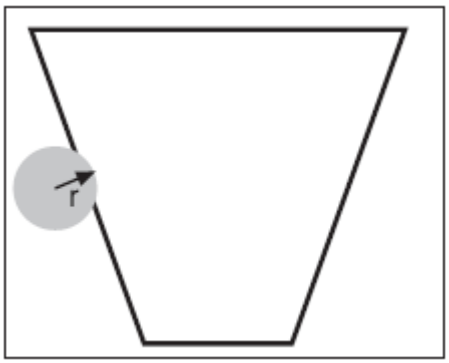
W tym przypadku P definiowane jest jako matryca projekcji, a C jest definiowane jako matryca kamery, która konwertuje obiekt z przestrzeni świata na przestrzeń kamery. Po zakończeniu konwersji masz do czynienia z prostym pudełkiem i nową płaszczyzną. Od tego momentu możesz obliczyć odległość między płaszczyzną a skrzynką. Jeśli jest przecięcie, płaszczyzna jest potencjalnie widoczna. Jeśli nie, powinieneś o tym zapomnieć.

Test Kula-Bryła Ścięta

Teraz, gdy rozumiesz testy przecięć, które zostały opracowane w poprzedniej części, takie problemy powinny być łatwe do rozwiązania. Strzała opisana jest za pomocą sześciu płaszczyzn, wszystkie skierowane są w stronę środka stożka ściętego. Najbardziej oczywistym testem, jaki można wykonać, jest to, czy środek kuli znajduje się po wewnętrznej stronie płaszczyzny. Przypomnij sobie, że możesz to osiągnąć, po prostu obliczając cztero-wektorowy produkt kropki między płaszczyzną a pozycją. Znak dodatni wskazuje, że wierzchołek znajduje się po właściwej stronie, a znak ujemny wskazuje, że wierzchołek nie znajduje się w obrębie ściętego stożka. (To nie powinno być dla ciebie nowym pojęciem, ponieważ zostało już użyte w części 4. Jeśli wszystkie sześć samolotów przejdzie test, twój obiekt jest potencjalnie widoczny i powinieneś go renderować. Znaczenie "potencjalnie" będzie wyjaśnione później W związku z tym test jest prosty: przejdź przez wszystkie sześć płaszczyzn $<n_p, D>$ stożka ściętego i sprawdź, czy następujący test przechodzi dla środka sfery p:

$$0 \le \langle \mathbf{n}, D \rangle \bullet \langle \mathbf{p}, 1 \rangle$$

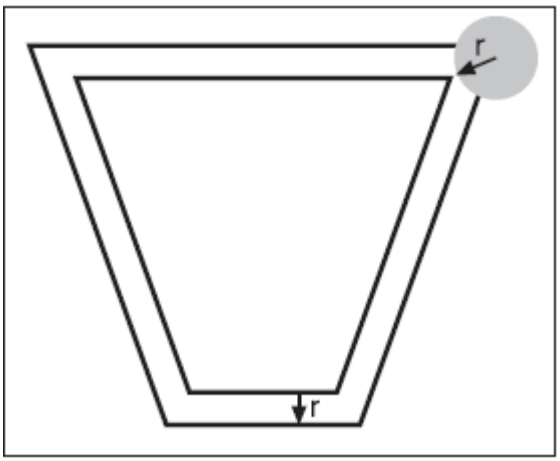
Gdybyś mógł użyć tylko tego jako testu, byłoby wspaniale. Obecnie jednak nie może to być kompleksowy test. Brakuje niektórych przypadków, w których kulka jest widoczna, ale nie jest wykrywana, jak pokazano na rysunku 15.2



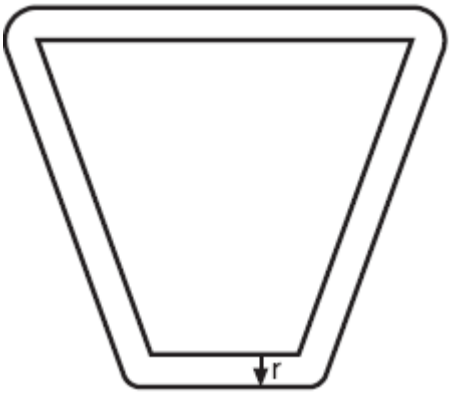
Problem polega na tym, że zajmujesz się sferą, a nie punktem w przestrzeni. Aby ten test był kompleksowy, możesz wypchnąć wszystkie sześć płaszczyzn z centrum środka ścięgnistego wzdłuż normalnej płaszczyzny z odległością promienia r. Mówiąc w skrócie, wydłuża się ścięgno tak, że każda jego płaszczyzna wzrasta o r. To rzeczywiście wykryje każdy przypadek, w którym sfera jest widoczna. Nowe równanie w celu osiągnięcia tego sprowadza się do następujących dla znormalizowanego normalnego n:

$$-r \le \langle \mathbf{n}, D \rangle \bullet \langle \mathbf{p}, 1 \rangle$$

Podczas przechodzenia przez wszystkie sześć płaszczyzn, wychodzisz, gdy test się nie powiedzie. Ten test jest dość prosty i szczupły, ale czy wszystko doskonale sobie z nim radzi? Nie całkiem. Wykryje każdy potencjalnie widoczny obiekt, a także zaznaczy obiekty, które nie są widoczne, jak pokazano na rysunku 15.3.



Aby być matematycznie poprawnym, idealny test musiałby zwiększyć promień sfery za pomocą promienia kuli w każdym punkcie cielska. W tym przypadku samoloty są po prostu przedłużane przez stożek ścięty. Gwarantuje to "idealne" dopasowanie wzdłuż krawędzi płaszczyzn, ale nie sprawdza się prawidłowo w rogach ścięgna, które powinny być rozciągnięte na kulki, jak pokazano na rysunku 15.4.



Na szczęście nie jest to wielka sprawa, jeśli wykryjesz, że niektóre obiekty są widoczne, gdy tak naprawdę nie są. Nie stanie się to zbyt często, ponieważ dwa wydłużone stożki ścięte na rycinach 15.3 i 15.4 różnią się tylko ośmioma małymi obszarami w narożnikach. Jeśli chodzi o szybkość, jaką można uzyskać z tego testu, to jest tego warte.

Test Pole-Bryła Ścięta

Jeśli przeprowadzisz szybkie wyszukiwanie w Internecie w celu rozwiązania tego problemu, typowe rozwiązanie pokazuje, jak deweloperzy nie myślą w kategoriach wektorów (co jest złe). Typowym rozwiązaniem jest rozważenie każdego rogu pudła jako oddzielnego wierzchołka i sprawdzenie, czy którykolwiek z wierzchołków pudła znajduje się po dodatniej stronie płaszczyzny. Jeśli istnieje wierzchołek dla każdego z płaszczyzn ściętego stożka, to spełnia ten test i pole jest widoczne. Wiąże się to łącznie z 36 testami dla pojedynczego pola, co oznacza 36 iloczynów wektorowych i nieefektywny test. Zamiast tego należy skorzystać z rozwiązania zaproponowanego w ostatnim rozdziale, w którym należy obliczyć efektywny promień pudła nadanego w płaszczyźnie. Wymaga to tylko sześciu testów, tak jak zrobiono to ze sferą. Jedynym dodatkowym obliczeniem jest promień pudełka, który obejmuje dodatkowy produkt punktowy na test. Możesz zredukować moc obliczeniową z 36 iloczynów wektorowych do 12 iloczynów skalarnych, a od 35 warunków do zaledwie sześciu. Kiedy pracujesz z wektorami, pracujesz z elementami geometrycznymi. Zazwyczaj o wiele łatwiej jest spojrzeć na problem z tego punktu widzenia niż w kategoriach kartezjańskich jednostek <x, y, z>. Podsumowując,

test przebiega w następujący sposób dla każdej płaszczyzny bryły <n, d> i ramki o zakresie {x, y, z} i środku p:

$$-r_e = -\left|\mathbf{n}\bullet\mathbf{x}\right| - \left|\mathbf{n}\bullet\mathbf{y}\right| - \left|\mathbf{n}\bullet\mathbf{z}\right| \le \left\langle \mathbf{n}, d\right\rangle \bullet \left\langle \mathbf{p}, 1\right\rangle$$

Ponownie, ten test cierpi na te same problemy, które ilustruje test przecięcia kuli i bryły ściętej. Czasami pole zostanie wykryte jako widoczne, gdy tak naprawdę nie jest. W takim przypadku skrzynka wydłuży ścięgno. W pierwszym przypadku rozszerzony sferyczny ścisk został uzyskany przez "zwijanie kuli" wzdłuż krawędzi stożka ściętego. To samo dotyczy. Możesz przesuwać pudło tak, że maksymalizuje zasięg ściętego owocu, aby uzyskać "prawdziwe" region, w którym test byłby idealny do określenia, które pudełka znajdują się w ciele. Ponownie, ta krótka strata nie jest bardzo znacząca, a prędkość testu naprawdę przewyższa korzyści w 100% dokładne rozwiązanie. Duży błąd może zostać naliczony, jeśli jeden z kierunkowych wektorów skrzynki jest znacznie większy niż pozostałe. W takim przypadku błąd w rogach stożka ściętego staje się większy. W tym przypadku bardziej interesujące może być przybliżenie kształtu do innego, na przykład do cylindra.

Elipsoida-Bryła Ścięta

W poprzedniej części szybko wspomniano, że można wykonać test elipsoidalny, przekształcając przestrzeń w taki sposób, że elipsoida staje się kulą, a następnie anulować transformację. To nie jest złe rozwiązanie, a jeśli rozwiniesz matematykę dotyczącą rozwiązania i zastosujesz ją do skrzynki z elipsoidalną obudową, otrzymasz dokładnie to samo rozwiązanie, które zostanie tutaj uzyskane. Jedyna różnica polega na tym, że w tym przypadku podejmiesz inne podejście i spojrzysz na problem. Po pierwsze, powinieneś zacząć od przeanalizowania, co zostało zrobione w przypadku przecięcia w płaszczyźnie ramki. Rzut pudła na normalny wektor płaszczyzny może faktycznie być postrzegany jako składowa x, y i z wektora. Innymi słowy, długość rzutu dla wektora x pola może odpowiadać składowej x, podobnie dla y i z. W końcu pole jest opisane trzema ortogonalnymi osiami, co oznacza, że iloczynem punktowym każdego pola jest zero. Kiedy rozwiązujesz te problemy, w zasadzie mapujesz wektor <x, y, z> w długość. Oczywiście ten wektor x, y, z jest w rzeczywistości projektowaną długością:

$$\left\langle x, y, z\right\rangle = \left\langle \mathbf{n}\bullet\mathbf{x}, \mathbf{n}\bullet\mathbf{y}, \mathbf{n}\bullet\mathbf{z}\right\rangle$$

Mapowanie, które wybierzesz, mapuje wektor na odległość, tak aby każdy wektor z równymi normami tworzył kształt. Jeśli weźmiesz wszystkie wektory, które mają normę jednego dla 2-normy, zauważysz, że zestaw punktów, które spełniają ten warunek, tworzy krąg, jak pokazano w Części 2. Ciekawe jest to, że jeśli wybierzesz nieskończoność -norm, masz kwadrat. Golly, równanie norm nieskończoności jest takie samo, jak równanie efektywnego promienia pudła. Co za przypadek. Nieważne, że równanie dla przecięcia na planie prostokąta rzeczywiście oblicza normę nieskończoności twojego rzutowanego wektora. Ta norma jest idealnie dobrana do obliczania promienia pola ze względu na jego gwałtowne zachowanie kwadratowe. To wszystko jest miłe i pokazuje związek pomiędzy pudełkiem a normą nieskończoności, ale jak może pomóc rozwiązać problem elipsy? Cóż, odpowiedź jest już przed tobą. Biorąc pod uwagę wektor długości projektu, jeśli możesz odwzorować każdy wektor na powierzchni pudełka (lub pudełko, które jest naprawdę jak kulka z normą nieskończoności) na równą długość, powinieneś być w stanie znaleźć normę, która mapuje każdy punkt na powierzchni elipsoidy do równej długości. To jest tylko 2-norma. 2-norma odwzorowuje odległość euklidesową między punktem i środkiem, a tym samym mapuje dowolny punkt na powierzchni elipsoidy w równej długości. To jest tylko 2-norma. 2-norma mapuje odległość euklidesową pomiędzy punkt i środek, a tym samym mapuje

dowolny punkt na powierzchni elipsoidy w równej długości. W związku z tym można obliczyć efektywny promień elipsoidalny, stosując następującą formułę:

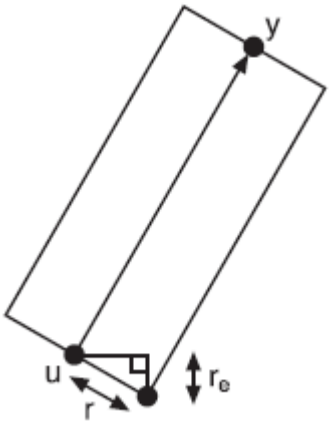
$$r_e = \sqrt{(\mathbf{n} \bullet \mathbf{x})^2 + (\mathbf{n} \bullet \mathbf{y})^2 + (\mathbf{n} \bullet \mathbf{z})^2}$$

Cylinder-Bryła Ścięta

Jeśli planujesz zbliżyć postać używając kapsuły lub cylindra, powinieneś również wiedzieć, jak sprawdzić, czy gracz jest widoczny, czy nie. W poprzednich testach wzór był prawie taki sam. Zacząłeś od obliczenia promienia obiektu (jeśli jeszcze go nie masz). Kiedy już miałeś promień, kontynuowałeś przedłużając ścięty promień o ten promień. Od tego momentu udało ci się z powodzeniem przekształcić problem obiektowo-bryłowy w problem z punktowo-bryłowy, co jest dość proste. Podobnie jak w przypadku cylindra, możesz przekształcić problem z obiektowo-bryłowy w problem punkt – bryła ścięta. W ten sposób definiujesz cylinder z dwoma punktami końcowymi: u i v oraz promieniem r. Możesz dalej zdefiniować wektor w jako wektor jednostkowy, który przechodzi od u do v:

$$w = \frac{\mathbf{u} - \mathbf{v}}{\|\mathbf{u} - \mathbf{v}\|_2}$$

Pierwszym krokiem do rozwiązania problemu jest znalezienie skutecznego promienia. Rysunek 15.5



ilustruje zależność między promieniem a różnymi składnikami cylindra. Po dokładnym zbadaniu i prostym powiązaniu trygonalnym, otrzymujesz następującą formułę dla efektywnego promienia:

$$\begin{aligned} r_e &= r\sin(\theta) \\ &= r\sqrt{1-\cos^2(\theta)} \\ &= r\sqrt{1-(\mathbf{v}-\mathbf{u})\bullet\mathbf{n}} \end{aligned}$$

Teraz, gdy masz już skuteczny promień, możesz wydłużyć płaszczyzny na zewnątrz o tę wartość. Prawdziwym problemem jest teraz dowiedzieć się, czy istnieje przecięcie między linią a płaszczyzną. Brzmi znajomo, prawda? Powinno! Najpierw musisz sprawdzić, czy każdy punkt końcowy znajduje się po innej stronie płaszczyzny, jak to zrobiono w części 4. Powiedzmy, że oba punkty końcowe podążają za tym warunkiem:

$$-r_e \geq \mathbf{n} \bullet \mathbf{a}$$

Tutaj a jest punktem końcowym {u, v}, a n jest normalne dla płaszczyzny. Możesz wywnioskować, że cylinder nie jest w ogóle widoczny. Z drugiej strony, jeśli stwierdzisz, że jeden lub oba punkty końcowe nie spełniają warunku, nie możesz niczego wydedukować i musisz zweryfikować następny samolot.

Generowania Ograniczonej Objętości

Do tej pory nauczyłeś się wykonywać wiele testów na kulach, cylindrach, kapsułach, pudełkach itp. W świecie rzeczywistym obiekty nie są wykonane z kul, cylindrów lub kapsułek. Przez większość czasu są one wykonane z trójkątów. Jak przydatne są te testy? Jak się okazuje, generalnie nie wykonasz testu przecięcia trójkąt-trójkąt. Zamiast tego, w końcu robisz budowanie obszaru ograniczającego, któremu gwarantowany jest zbiór wierzchołków. To znacznie przyspiesza obliczenia, ponieważ zasadniczo oblicza się kolizję wielu obiektów w jednym ujęciu. Ma on złą stronę, co oznacza, że jeśli skrzyżowanie zostanie wykryte z objętością ograniczającą, obiekty wewnętrzne prawdopodobnie nie kolidują ze sobą. W takim przypadku można albo stwierdzić kolizję z obiektem, albo zestrzelić i zabrudzić, aby dokładnie przeanalizować, który trójkąt obiektu został naruszony. Następna część skoncentruje się na takich rzeczach. Na razie skoncentrujmy się na problemie, który ma na celu wygenerowanie objętości ograniczającej w pierwszej kolejności.

Generowanie ograniczających pól

Jeśli weźmiesz prosty przypadek, w którym pole jest wyrównane względem osi, bardzo łatwo jest określić obwiednię dla zestawu wierzchołków. Wszystko, co musisz zrobić, to znaleźć minimalną i maksymalną wartość w {x, y, z}. Środek pola staje się środkiem tego, a zasięg pola staje się o połowę różnicy między wartościami max i min {x, y, z}. Jest całkiem oczywiste, że to pole łączy każdy wierzchołek, ponieważ skonstruowałeś go, przyjmując min / max w każdej osi. Teraz, jeśli chcesz zbudować pudło ogólne, sprawy stają się nieco bardziej skomplikowane. Najpierw musisz określić orientację pudełka. Tutaj nie ma znaczenia, czy proces ten zajmuje dużo czasu, ponieważ wystarczy to zrobić tylko raz. Ale kiedy już to zrobi, powinno się to zrobić dobrze. Najpierw wszystko. Zacznijmy od wybrania centrum dla twojego pudełka. Ponieważ pudełko nie jest wyrównane pod względem osi, robi się trochę skomplikowane. To tutaj przydaje się wszystkie pozornie bezużyteczne rzeczy dotyczące diagonalizacji z Części 3. Każda objętość ograniczająca wymaga obliczenia podstawy diagonalizacji, która jest rzeczywiście podstawą dla kierunku objętości. Jak widać w tej części, nie ma dużej różnicy między elipsoidą, pudełkiem i diamentem. Wszystko zależy od wybranej przez ciebie normy, więc proces jest bardzo podobny. Podstawy znalezione w procesie ortogonalnym powinny zostać znormalizowane. Rzeczywista długość wektorów tworzących podstawę {x, y, z} nie jest dla ciebie użyteczna. Ponieważ długość jest bez znaczenia, musisz samodzielnie obliczyć zasięg (odległość wzdłuż osi bazowych). Jest to łatwe, jeśli rzutujesz wektory na wektory bazowe. Daje to długość tego punktu od środka, a wybór minimum i maksimum iloczyn skalarny daje twoje samoloty. Dokładniej, osiem płaszczyzn, które wiążą skrzynię, można zdefiniować matematycznie w następujący sposób:

$$\left\langle \mathbf{x}, -\min_{1\leq i\leq n}\{\mathbf{v}_i \bullet \mathbf{x}\} \right\rangle \quad \left\langle -\mathbf{x}, \max_{1\leq i\leq n}\{\mathbf{v}_i \bullet \mathbf{x}\} \right\rangle$$
$$\left\langle \mathbf{y}, -\min_{1\leq i\leq n}\{\mathbf{v}_i \bullet \mathbf{y}\} \right\rangle \quad \left\langle -\mathbf{y}, \max_{1\leq i\leq n}\{\mathbf{v}_i \bullet \mathbf{y}\} \right\rangle$$
$$\left\langle \mathbf{z}, -\min_{1\leq i\leq n}\{\mathbf{v}_i \bullet \mathbf{z}\} \right\rangle \quad \left\langle -\mathbf{z}, \max_{1\leq i\leq n}\{\mathbf{v}_i \bullet \mathbf{z}\} \right\rangle$$

To świetnie, ale w poprzednich przykładach pracowałeś z dużymi i środkowymi polami. Łatwo jest konwertować z jednego na drugi. Aby przekształcić ten zestaw płaszczyzn w centrum, wystarczy wziąć średnią z dwóch płaszczyzn. Wiesz, że całkowity rozmiar pudełka to tak naprawdę maksymalne przesunięcie minus min. Przesunięcie. W ten sposób obliczasz prostą długość. Jeśli masz dwie wartości i chcesz poznać normę, bierzesz większą z dwóch wartości i odejmujesz ją od mniejszej. To samo dzieje się tutaj. To przesunięcie daje przesunięcie (lub zakres) dla tego konkretnego wektora kierunkowego. Chcesz połowy tego całkowitego zakresu, ponieważ obliczasz płaszczyzny jako jeden wektor kierunkowy, w którym znajduje się jedna płaszczyzna, dodając środek do tego wektora kierunkowego, a drugi, odejmując wektor kierunkowy od środka. W związku z tym potrzebujesz połowy całkowitego zasięgu. Matematyczne równanie dla zakresów {| x |, | y |, | z |} jest po prostu podane w następujący sposób:

$$|\mathbf{x}| = \frac{\max\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{x}\} + \min\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{x}\}}{2}$$

$$|\mathbf{y}| = \frac{\max\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{y}\} + \min\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{y}\}}{2}$$

$$|\mathbf{z}| = \frac{\max\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{z}\} + \min\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{z}\}}{2}$$

Zwróć uwagę, że tutaj zamiast notacji normalnej używana jest wartość bezwzględna. To głównie normą x jest pełny zakres, a tutaj bezwzględna wartość x jest określona jako pół-zakres lub zasięg pola od środka. Podobnie dla y i z. Tak więc wszystko, co musisz zrobić, aby znaleźć środek, to przesuwać się wzdłuż wektora kierunkowego pół-zakresu. Te połowy zostały już wyliczone powyżej, więc obliczenie środka pudełka jest dość łatwe. Matematycznie, patrząc na następujące równanie dla środka pola c:

$$\mathbf{c} = \frac{\max\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{x}\} + \min\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{x}\}}{2}\mathbf{x} + \frac{\max\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{y}\} + \min\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{y}\}}{2}\mathbf{y} + \frac{\max\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{z}\} + \min\limits_{1\le i\le n}\{\mathbf{v}_i \bullet \mathbf{z}\}}{2}\mathbf{z}$$

Generowanie ograniczone kuli

Jak widzisz, kule tworzą całkiem skuteczny sposób obliczania kolizji lub stosowania testu uśmiercania, więc spójrzmy, jak skonstruować graniczną sferę dla zestawu wierzchołków. Technika opiera się na tych samych zasadach, które zostały opracowane powyżej. Kule są inne niż pudełka, ponieważ nie masz kontroli nad zasięgiem w trzech różnych kierunkach. Za pomocą tego pola znajdujesz orientację wierzchołków, obliczając wektory własne. Daje to ogólny kierunek pudła i ogólne poczucie kierunku (większa norma oznacza większy zakres w stosunku do tego wierzchołka), ale nie zapewnia w sposób dorozumiany zakresu. Musisz sam obliczyć zasięg, znajdując minimalny i maksymalny wierzchołek w tym kierunku. Ponieważ jest to pudło, budowanie płaszczyzny przy min i max gwarantuje, że wszystkie wierzchołki znajdują się po jednej stronie płaszczyzny. Gdyby to nie było prawdą, nie byłyby one min i max, prawda? W przypadku kuli nie można ustawić promienia i zagwarantować, że każdy wierzchołek znajduje się w sferze. Jeszcze większym problemem jest to, że masz trzy osie, ale sfera ma tylko jeden parametr oprócz jej środka: promień. Dlatego musisz wybrać największą z trzech osi. Największy stopień (lub główna oś) odpowiada największej uzyskanej wartości własnej. Jak wspomniano wcześniej, wartość własna daje poczucie względnej normy, ale nie daje rzeczywistego zasięgu. Odtąd niech wektor odpowiadający największej wartości własnej będzie q.

Uwaga

Jeśli macierz M jest symetryczna, największa niepodpisana wartość własna jest równa 2-normie M. Bardziej ogólnie, macierz 2-normalna może być obliczona jako pierwiastek kwadratowy z większej wartości własnej dla MMT.

Teraz, gdy znasz najważniejszy kierunek dla zestawu punktów, po prostu potrzebujesz znaleźć zasięg wierzchołków w tym kierunku. Odbywa się to dokładnie tak, jak wcześniej. Jeśli obliczysz rzut każdego wierzchołka na q, możesz uzyskać wartości min i max, które tworzą zasięg w tym kierunku. Tak więc, w kategoriach równań, masz następujące:

$$\mathbf{u} = \left\{ \mathbf{p}_i \bullet \mathbf{q} = \max_{1 \le i \le n} \left\{ \mathbf{p}_i \bullet \mathbf{q} \right\} \quad \mathbf{p}_i \right\}$$
$$\mathbf{v} = \left\{ \mathbf{p}_i \bullet \mathbf{q} = \min_{1 \le i \le n} \left\{ \mathbf{p}_i \bullet \mathbf{q} \right\} \quad \mathbf{p}_i \right\}$$

I oczywiście, średnia między tymi dwoma punktami, u i v, da ci centrum c sfery. Tymczasem połowa odległości między tymi dwoma wierzchołkami da ci promień:

$$\mathbf{c} = \frac{\mathbf{u} + \mathbf{v}}{2}$$
$$r = \left\| \mathbf{u} - \mathbf{c} \right\|_2$$

Niestety, z samej natury konstrukcji pola, masz gwarancję, że każdy wierzchołek jest zamknięty w pudełku. Nie dotyczy to sfery. To tylko przybliżenie realnej oferty. Generowanie granicznych sfer (a nawet granicznych kręgów) nie jest banalnym problemem, a obecnie nie ma czystego, niezawodnego rozwiązania. Ta metoda daje całkiem przyzwoite przybliżenie, bez konieczności starzenia. Objętość ograniczająca, która nie ogranicza całego woluminu, jest całkiem bezużyteczna, więc przynajmniej powinieneś mieć sposób na obliczenie innej granicznej sfery, która obsługuje każdy wierzchołek. Jednym ze sposobów, w jaki można to osiągnąć, jest sprawdzenie, czy odległość między wierzchołkiem a środkiem kuli jest mniejsza lub równa promieniu:

$$\left\| \mathbf{p}_i - \mathbf{c} \right\|^2 \le r^2$$

Krótko mówiąc, sprawdzasz, czy wierzchołek znajduje się w sferze. Jeśli tak nie jest, można zmienić promień tak, że otaczająca kula obejmuje wierzchołek. Sam w sobie nie jest to złe rozwiązanie, ale dodaje dodatkową przestrzeń po drugiej stronie kuli, gdzie nie może być żadnych wierzchołków. Oznacza to, że faktycznie obliczysz sferę, która jest większa niż to konieczne. Zamiast tego, możesz rozszerzyć kulę o promień mniejszy niż różnica między promieniem a długością i przesunąć kulkę tak, aby była nieznacznie w kierunku wierzchołka, aby skompensować przyrost mniejszego promienia. Ta technika gwarantuje, że każdy wierzchołek znajdujący się wewnątrz objętości granicznej nadal znajduje się wewnątrz sfery granicznej, a także minimalizuje rozmiar zwiększenia promienia. Więc jaka jest magiczna sztuczka, aby to zrobić? Musisz znaleźć pozycję, w której przesuniesz środek kuli. Jeśli zbliżysz się do wierzchołka, który nie był ograniczony przez kulę, możesz zwiększyć promień na tyle, aby sfera była gwarantowana, aby związać obszar, który wcześniej ograniczał. Jest to konieczne, bo inaczej po prostu przestaniesz ścigać swój ogon. Możesz łatwo znaleźć wierzchołek na drugim końcu sfery i w

przeciwnym kierunku qp_i (wektor zaczyna się od q do p_i). Ponieważ punkt ten znajduje się na kuli, łatwo jest znaleźć jego położenie. Niech ten punkt będzie w:

$$\mathbf{w} = \mathbf{c} - r\frac{\mathbf{p}_i - \mathbf{c}}{\left\|\mathbf{p}_i - \mathbf{c}\right\|_2}$$

Od tego momentu jest to całkiem proste. Masz dwa przeciwne ekstrema na a kula, więc środek kuli znajduje się w średniej z tych dwóch punktów, a nowy promień stanowi połowę odległości między tymi dwoma punktami .Matematycznie:

$$\mathbf{c}' = \frac{\mathbf{w} + \mathbf{p}_i}{2}$$
$$r' = \left\|\mathbf{c}' - \mathbf{p}_i\right\|_2$$

Jeśli powtórzysz ten proces, aż każdy pojedynczy wierzchołek na liście znajduje się wewnątrz obwiedni kuli, skonstruujesz kulę, która łączy każdy pojedynczy wierzchołek na twojej liście. To nie jest minimalna sfera graniczna, ale jest bardzo blisko.

Generowanie Ograniczającej Elipsoidy

Jak zauważono, elipsoida jest ściśle związana z kulą. To tylko kula, która została nierówno skalowana na trzech różnych osiach. Jeśli pamiętasz o tym, znalezienie granicznej elipsoidy przypomina znalezienie sfery granicznej. Ponownie, pierwszym krokiem jest obliczenie wektorów kierunkowych {x, y, z} dla zbioru wierzchołków. Kiedy już to zrobisz, powinieneś również obliczyć stopień, jaki został zrobiony w dwóch poprzednich metodach. W tym miejscu zdefiniuj długość zakresów w następujący sposób:

$$a = \max_{1\le i\le n}\{\mathbf{p}_i \bullet \mathbf{x}\} - \min_{1\le i\le n}\{\mathbf{p}_i \bullet \mathbf{x}\}$$
$$b = \max_{1\le i\le n}\{\mathbf{p}_i \bullet \mathbf{y}\} - \min_{1\le i\le n}\{\mathbf{p}_i \bullet \mathbf{y}\}$$
$$c = \max_{1\le i\le n}\{\mathbf{p}_i \bullet \mathbf{z}\} - \min_{1\le i\le n}\{\mathbf{p}_i \bullet \mathbf{z}\}$$

Jeśli przypomnisz sobie, w jaki sposób uzyskałeś kierunek zasięgu, rozumiesz, jak przekształcić każdy punkt w przestrzeń, w której elipsoida staje się kulą. Po pierwsze, jeśli chcesz zamienić elipsoidę ustawioną w osi na sferę, wszystko co musisz zrobić, to przeskalować oś wyrównaną elipsoidę przez długość promienia każdej osi. Innymi słowy, elipsoida o promieniu {a, b, c} dla {x, y, z} musi zostać odpowiednio zmniejszona o {1 / a, 1 / b, 1 / c}. Tak więc matrycą, która skaluje tę specyficzną elipsoidę w kulę jest K:

$$\mathbf{K} = \begin{bmatrix} \frac{1}{a} & 0 & 0 \\ 0 & \frac{1}{b} & 0 \\ 0 & 0 & \frac{1}{c} \end{bmatrix}$$

Kiedy równanie, którego użyłeś do transformacji punktów i uzyskasz oś kierunkową dla objętości granicznej, zostanie zastosowane do K, daje ona następujące wartości:

$$\mathbf{M} = \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{bmatrix} \mathbf{K} \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{bmatrix}^{\mathrm{T}}$$

$$= \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{bmatrix} \begin{bmatrix} \frac{1}{a} & 0 & 0 \\ 0 & \frac{1}{b} & 0 \\ 0 & 0 & \frac{1}{c} \end{bmatrix} \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{bmatrix}^{\mathrm{T}}$$

Pomysł jest więc następujący: musisz najpierw przekształcić punkty za pomocą macierzy M. Po transformacji każdego z tych punktów, należy wykonać obliczenia sfery granicznej, aby znaleźć środek sfery. Gdy już masz środek sfery, możesz usunąć transformację, stosując odwrotność M, którą łatwo uzyskać przez obserwację reguł macierzy:

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{bmatrix} \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{bmatrix}^{\mathrm{T}}$$

Same długości uzyskuje się poprzez przekształcenie promienia za pomocą K. Innymi słowy, wystarczy pomnożyć promień r uzyskany przez obliczenie objętości granicznej sfery przez zakres elipsoidy {a, b, c}. Kierunek elipsoidy wyznaczają wartości kierunkowych wektorów {x, y, z}. Pod względem równań:

$$a' = ra \quad \text{Along } \mathbf{x}$$
$$b' = rb \quad \text{Along } \mathbf{y}$$
$$c' = rc \quad \text{Along } \mathbf{z}$$

Ponownie, jest to tylko kwestia wiedzy o tym, w którym miejscu i układzie współrzędnych pracujesz. Kiedy się o tym dowiesz, wystarczy przekształcić jeden problem w prostszy problem, który wiesz, jak rozwiązać. Po rozwiązaniu tego problemu musisz odwrócić dokonaną konwersję, aby uzyskać oczekiwane wyniki. Jest to świetny sposób na rozwiązanie wielu problemów, ale działa tylko wtedy, gdy istnieje możliwość unikatowego odwrócenia zmiany, którą wykonałeś, aby przekonwertować problem na ten, który wiesz, jak rozwiązać.

Usuwanie Tylnej Ściany w 3D

Zasadniczo usunie tylnej ścianu nie jest popularną metodą, ponieważ sprzęt wykonuje już coś bardzo zbliżonego do tego. Jednak usunięcie tylnej ściany na poziomie procesora może być atrakcyjną opcją, jeśli masz ograniczoną przepustowość (liczba wierzchołków wysłanych do GPU). W takim przypadku możesz być zainteresowany obliczaniem, czy trójkąt lub czworokąt jest zwrócony w stronę kamery, czy też nie. Jedyną trudną rzeczą jest to, że nie masz współrzędnych 2D trójkąta / kwadratu. Istnieje jednak inna metoda, która pozwala określić, czy twarz jest widoczna, czy nie. W duchu określania wielokąta w sposób zgodny z ruchem wskazówek zegara / przeciwny, można obliczyć normalny wektor wielokąta. Ponownie, wektor normalny może mieć jeden z dwóch kierunków, które są zależne od kolejności

wierzchołków (zgodnie z ruchem wskazówek zegara lub nie). W 3D nie ma sensu po prostu patrzeć na znak składnika z, ponieważ nic nie gwarantuje, że jesteś wyrównany względem osi. W rzeczywistości jest to rzadkie. Zamiast tego podejście to analizuje kąt między wektorem kierunkowym kamery a normalnym wektorem płaszczyzny. Jeśli kąt między dwoma wektorami jest mniejszy niż 90 stopni, wektor normalny jest skierowany w stronę kamery, a zatem wielokąt, który ją reprezentuje, jest potencjalnie widoczny. Z drugiej strony, jeśli kąt jest większy niż 90 stopni, wektor normalny jest skierowany w tym samym kierunku co wektor kamery. W takim przypadku wielokąt nie jest widoczny. Na koniec, który test z Części 2 może pomóc w rozwiązaniu zagadki związanej z kątem 90 stopni? Oczywiście, jest to produkt z kropką. Iloczyn iloczynowy dwóch wektorów jest dodatni, gdy kąt między dwoma wektorami jest mniejszy niż 90 stopni, a w przeciwnym razie jest ujemny. Tak więc test jest dość prosty. Biorąc pod uwagę wektor kierunkowy kamery c oraz wektor normalny wielokąta n, można obliczyć jego widoczność w następujący sposób:

$$\mathbf{c} \bullet \mathbf{n} = \begin{cases} \geq 0 & \text{Widoczne} \\ < 0 & \text{Niewidoczne} \end{cases}$$

Wyłapywanie obiektów

Więc ograniczyłeś swoje kształty, aby określić, czy były widoczne na ekranie, usunięto tylne ściany procesora, a mimo to nadal potrzebujesz dodatkowej przepustowości GPU i wciąż masz przepustowość procesora. Gdy wykryjesz obiekt / wielokąt jak widać po poprzednich testach, nie gwarantuje, że jest naprawdę widoczny. W związku z tym jest potencjalnie widoczny. Na przykład, jeśli spojrzysz w sufit swojego domu, czy widzisz niebo? Nie, chyba że masz dach nad głową. A niebo wciąż jest w twoim widoku. Gdyby nie było dachu, zobaczyłbyś niebo. Nie widzisz nieba, ponieważ twój dach go zasłania. Pomysł polegający na usuwaniu obiektów polega na określeniu, czy niektóre obiekty ukrywają inne obiekty, a jeśli tak, całkowicie usuwają ukryte obiekty. Oznacza to, że możesz usuwać całe obiekty bez konieczności dokładniejszego ich przeglądania. W tym momencie powinieneś mieć tylko zestaw obiektów, które znajdują się w ciele ściętym. Aby ten test był w ogóle przydatny, musi istnieć zestaw dobrych okludatorów w twoim świecie. Zdefiniujmy termin "dobry okludator". Jergo kandydata do okluzji. W grze nie ma wielu dużych obiektów, które mogłyby zasłaniać widok twojej postaci. (Mury się nie liczą, ponieważ składają się z bardziej złożonej struktury, którą zobaczysz w następnym rozdziale). Z drugiej strony, jeśli weźmiesz grę, taką jak Grand Theft Auto 3, gdzie jest dużo w małych budynkach, na wysokim terenie, niskim terenie, samochodach i podobnych obiektach istnieje wiele potencjalnie dobrych okludatorów. Ale znowu ściany budynku tak naprawdę się nie liczą. Patrzysz raczej na duży obiekt, który kryje za sobą wiele świata. Inną rzeczą, na którą musisz uważać, jest to, że ta metoda powinna być stosowana tylko wtedy, gdy zawsze jest okludator nie za daleko. Jeśli tak nie jest, typowy gracz oczekiwałby, że gra będzie względnie szybka przez cały czas. Jeśli gra jest szybka tylko wtedy, gdy idziesz przed wzniesieniem, wdrożenie tej metody jest prawie bezużyteczne. Zamiast tego powinieneś popatrzeć na lepszą technikę podziału, co zostanie pokazane w następnym rozdziale. Tak więc idealnym rozwiązaniem dla okluzji jest sytuacja, w której prawie cały czas zasłania wiele obiektów. Kolejnym dobrym tego przykładem jest konkretny poziom w Unreal Tournament. Ten poziom ma wiele wzgórz. Wzgórza są dużymi obiektami, które są prawie wszędzie, więc są dobre okludatory. Jeśli postać wspina się na wzgórze, nadal będziesz potrzebować odpowiedniej liczby klatek na sekundę, która naprawdę zależy od rodzaju gry, którą tworzysz. Ale jeśli postać na ogół nie będzie na szczycie wzgórza, które znajduje się na szczycie świata (lub może zobaczyć większość świata), jesteś bezpieczny. Tak to mówi ci kiedy użyć tej techniki, ale czym właściwie jest ta technika? To pole jest całkiem nowe i nie wykonano zbyt wiele pracy. Często deweloperzy boją się wprowadzić takie techniki, ponieważ przyrosty prędkości nie są jasne, ale ta może być bardzo korzystna w odpowiedniej sytuacji. Z drugiej

strony, jeśli zastosujesz tę technikę i pojawi się kilka nielicznych okładatorów, będziesz się kopał w głowę. Nie ma magicznego równania, aby znaleźć najlepszego okludera. Wykorzystaj swoją ocenę do określonej gry. Zasadniczo odległość kamery od okludera i objętość pobierana przez okluder odgrywają dużą rolę w tym, czy weryfikacja okluzji przeciwko temu obiektowi byłaby korzystna czy nie. Przez większość czasu prawdopodobnie można uciec z obszarem ograniczonej głośności obiektu i odległości między obiektem a kamerą. Możesz wymyślić równanie, które daje zakres akceptowalności. Na przykład możesz twierdzić, że jakakolwiek liczba z przedziału [0..1] oznacza, że obiekt powinien być uwzględniany w okluzji, podczas gdy wszystko większe niż to nie powinno być brane pod uwagę. Na przykład, oto taka funkcja:

$p = Odległość^2 / Obszar * 10$

Trzy rzeczy mogą to zrobić. Po pierwsze, obiekt musi być naprawdę blisko, w którym to przypadku p staje się naprawdę mały. Inną możliwością jest to, że obiekt jest dość oddalony od widoku, ale jest tak duży, że wciąż wydaje się duży. Ostatnią opcją jest dobrze wyważona sytuacja, w której obiekt jest względnie blisko i o rozsądnej wielkości. W tym przypadku stosuje się współczynnik 1/10 (należy go dostosować do konkretnej sytuacji w grze). Inną interesującą częścią jest odległość do kwadratu. Dzieje się tak, ponieważ odległość ma całkiem istotny wpływ na obszar, w którym obiekt zanika z ekranu. Na przykład weź książkę i umieść ją na swoich oczach, abyś nie widział niczego innego. Teraz przenieś książkę z powrotem, aż twoje ramię zostanie całkowicie wysunięte. Teraz, gdy książka jest dalej, prawdopodobnie zatyka 2-3 średnie obiekty. Gdy była blisko, ukryła prawie wszystko. Teraz przejdźmy do prawdziwego problemu, który w rzeczywistości usuwa okludowane obiekty.

Wypukła okluzja cienia

Ta technika rozwija pomysł, że można wydzielić zbiór obiektów, jeśli można zbudować cień obiektu zamykającego. To ma sens, jeśli myślisz o rozwiązaniu w kategoriach świata. Weź źródło światła i umieść je w centrum swojego świata. Teraz weź kilka wybranych okluderami i obserwuj, jak cienie są rzucane na różne inne obiekty za okluderami. Jeśli obiekt jest widoczny, światło świeci bezpośrednio na nim. Jeśli obiekt nie jest widoczny, światło nie jest rzucane na niego. Jeśli zastąpisz światło aparatem, dzieje się podobnie. Światło tylko oświetla to, co widać bezpośrednio ze źródła. Jeśli nie może zobaczyć obiektu, nie może go bezpośrednio oświetlić. Tak więc wszystko, co widzi kamera, jest odpowiednikiem tego, co świeci źródło światła. W związku z tym, jeśli można znaleźć objętość zacienionego regionu, można wykonać test przecięcia, aby sprawdzić, czy obwiednia obiektu jest całkowicie zacieniona. Jeśli tak, obiekt nie powinien być w ogóle rysowany. Jest to dość prosta zasada, ale łatwiej powiedzieć niż zrobić. W tym momencie powinieneś mieć wystarczającą ilość tła, aby określić, czy obiekt jest widoczny, czy nie, biorąc pod uwagę zestaw płaszczyzn. Ta część powinna być spacerkiem po parku. Nie jest tak jasne, jak zbudować zestaw samolotów, które tworzą zacieniony region. Pierwszą rzeczą, którą powinieneś zrobić, jest poziom grafiki. Dla każdego obiektu, który będzie potencjalnym okluderem, należy wygenerować dwie reprezentacje. Pierwsza reprezentacja jest tym, co gracz zobaczy na swoim ekranie. Jest to pełnoprawna, teksturowana i szczegółowa wersja obiektu. Druga reprezentacja jest wypukłą wersją obiektu, która jest całkowicie zawarta w poprzednim obiekcie. Potrzebujesz ciała wypukłego (objętość, która nie ma żadnych wgnieceń wewnętrznych), ponieważ znacznie łatwiej jest sprawdzić, czy obiekt znajduje się w tym woluminie, czy nie. Istnieje 5-6 algorytmów, które mogą budować wypukłe ciała z zestawu wierzchołków. Ten, który preferuję, to metoda pakowania prezentów, która działa jak pakowanie prezentu świątecznego. To konstruuje zbiór wierzchołków / płaszczyzn, które otaczają cały obiekt. Problem z tego typu algorytmem dla tego konkretnego rozwiązania polega na tym, że generuje on wolumen równy lub większy niż poprzedni obiekt. Nie można użyć większej głośności, bo w przeciwnym razie niektóre obiekty będą potencjalnie widoczne. W związku z tym należy uzyskać objętość wypukłą, która jest całkowicie zawarta w

poprzednim obiekcie. Krótko mówiąc, jeśli nałożycie dwa obiekty na siebie, oryginał i wypukłe ciało, wypukłe ciało powinno całkowicie zniknąć z widoku, ponieważ jest zawarte w pierwotnej objętości. Oczywiście można wygenerować taki algorytm, ale łatwiej i bardziej optymalnie go wygenerować ręcznie. Jest to optymalne, ponieważ dzięki temu, korzystając z narzędzia do modelowania 3D, można również usuwać detale z obiektu, które są bezużyteczne lub nieistotne dla celów uboju. Ponownie, jest to przewidziane, że obiekt jest całkowicie zawarty w drugim i jest wypukły. Na przykład, jeśli chcesz uzyskać obudowane wypukłe ciało budynku, możesz zapomnieć o dachu (który może wystawać z fundamentu), krawędziach stopni okna i fantazyjnych klockach, po prostu zastępując całą rzecz pudełko. To znacznie zmniejszy wymaganą moc obliczeniową. Po uzyskaniu tych dwóch obiektów, celem jest renderowanie oryginalnego "złożonego" obiektu i używanie uproszczonego obiektu do celów uboju. Ponieważ późniejsza objętość jest wypukła, możesz łatwo sprawdzić, czy cokolwiek znajduje się wewnątrz objętości, zastępując punkt równaniem płaszczyzny:

$$d = \langle \mathbf{n}, D \rangle \bullet \langle \mathbf{x}, 1 \rangle$$

Otrzymuje on płaszczyznę <n, D> i wierzchołek x. Znak będzie dodatni, jeśli jesteś na tej samej półpłaszczyźnie, że normalne punkty w kierunku, i odwrotnie, wynik ujemny oznacza, że wierzchołek jest umieszczony w drugiej połowie płaszczyzny. Jeśli każda płaszczyzna nie wskazuje wierzchołka (d = 0 dla normalnych wskazujących na zewnątrz), wiesz, że wierzchołek znajduje się wewnątrz wypukłego ciała. Teraz masz wypukłe ciało, ale nie masz cienia. Nie chcesz sprawdzać, czy obiekt znajduje się wewnątrz potencjalnego cullera, ale że obiekt znajduje się za nim. W związku z tym musisz obliczyć wypukłe ciało cienia, które wygląda jak stożek. Możesz łatwo określić zestaw samolotów, które pomogą ci to przetestować. Jeśli weźmiesz zestaw trójkątów, które są widoczne z punktu widzenia kamery, każdy z tych trójkątów znajduje się na granicy ciała wypukłego. Zatem wszystkie one powinny być częścią wypukłego ciała cienia. Te trójkąty mogą zostać przekształcone w płaszczyzny dla twojego testu ciałko-wypukły-punkt-punkt, widzianego wcześniej w tej sekcji. Chociaż te samoloty są liczne, nie są wystarczające, ponieważ zakrywają obiekty na granicy. Nakrywają tylko wypukłe ciało cienia z przodu. Musisz również zadbać o boki. Aby uzyskać płaszczyzny budujące przedłużenie cienia, należy najpierw określić, która krawędź wypukłego ciała definiuje granicę wypukłego ciała. Jeśli możesz znaleźć każdą krawędź, która definiuje granicę ciała wypukłego z danego punktu widzenia, możesz wygenerować płaszczyznę ze środka kamery i dwa wierzchołki zawierające krawędź, która odpowiada jednej części płaszczyzny bocznej cienia. Istnieje bardzo prosty sposób uzyskania tej listy krawędzi. Podczas obliczania, które trójkąty są widoczne, należy również śledzić krawędzie. Wstaw każdą krawędź dla każdego widocznego trójkąta do listy. Jeśli krawędź już istnieje na liście, usuń ją, ponieważ oznacza to, że istnieje inny widoczny trójkąt, który dzieli tę krawędź. Jeśli dwa trójkąty mają tę samą krawędź, to ta sama krawędź nie może zdefiniować granicy kształtu z punktu widzenia kamery. W końcu pozostają krawędzie, których nie dzielą żadne dwa widoczne trójkąty. Po uzyskaniu tego zestawu krawędzi można utworzyć zestaw płaszczyzn, które wraz z płaszczyznami widocznego trójkąta budują wypukły cień w cieniu. Aby być technicznie poprawne, ta objętość ograniczająca nie jest prawdziwie wypukła, ponieważ druga strona obiektu nie jest zamknięta. Jednak w twoim przypadku nie potrzebujesz, aby obiekt był zamknięty w tej konkretnej lokalizacji. Test wnętrza z wypukłym ciałem nadal będzie działał dobrze. Po zbudowaniu zestawu samolotów do przetestowania, możesz po prostu przetestować w stosunku do obwiedni obiektów, aby określić, czy te obiekty powinny zostać poddane ubojowi, czy nie. Przypomnijmy, że obiekt musi znajdować się całkowicie wewnątrz cienia, aby to zadziałało. Jeśli jakiś obiekt jest widoczny, powinien zostać narysowany. W związku z tym będziesz musiał zmodyfikować testy przecięcia dla sfery / ramki / etc. aby upewnić się, że cały obiekt znajduje się wewnątrz wypukłego ciała. Jest to całkiem proste, jeśli wszystko określisz, obliczając odległość między płaszczyzną a obwieszonym obiektem. Teraz prawdopodobnie możesz zobaczyć, dlaczego

ważne jest, aby najpierw utworzyć wypukłe ciało, które nie ma zbyt wielu szczegółów. Im więcej szczegółów dodasz do wypukłego ciała, tym bardziej szczegółowa staje się objętość cienia i tym więcej samolotów musisz przetestować.

Szczegóły Odrzucenia

Pomysł na utylizację szczegółów polega na tym, że obiekty znajdujące się stosunkowo daleko od kamery nie potrzebują tak bardzo szczegółów jak obiekty, które są naprawdę blisko aparatu. Jest to dość oczywiste. Jeśli masz niezwykle szczegółowego lwa i patrzysz na jego twarz z bliska, wygląda to całkiem dobrze. Jeśli weźmiesz tego samego lwa i odsuniesz go o blok, prawdopodobnie nie będziesz w stanie stwierdzić, jaki jest kolor jego oczu lub czy ma nawet wąsy. Taki jest cały pomysł za utylizacją detali. Jest to funkcja odległości. Im dalej znajduje się obiekt, tym mniej szczegółów potrzebujesz. Co więcej, im większy obiekt, tym więcej szczegółów jest ważnych. Jeśli masz tygrysa i mały kamień, oba znajdujące się blisko aparatu, prawdopodobnie nie dbasz o to, czy skała jest szczegółowa na śmierć, ponieważ tygrys zajmuje większość miejsca na ekranie. W tym przypadku najlepszym rozwiązaniem jest wdrożenie różnych wersji modeli dla różnych głębokości. Możesz rozdzielić głębokości w różnych interwałach, jeśli chcesz, i te interwały pokażą mniej szczegółowe obiekty wraz ze wzrostem głębokości. Liczba modeli, które przechowujesz zależy od tego, ile pamięci i przepustowości możesz sobie pozwolić. Istnieją również metody, które próbują dynamicznie usuwać trójkąty, gdy obiekt jest odsunięty od przeglądarki. Niefortunna prawda o tych metodach polega na tym, że nie są one bardzo wydajne i mają tendencję do generowania rezultatów, gdy trójkąty wydają się pojawiać znikąd. Oczywiście, ten efekt może się zdarzyć, jeśli wygenerujesz wiele różnorodnych szczegółowych modeli, ale można je kontrolować. Istnieją również metody, które próbują całkowicie usunąć ten efekt popping przez stopniowe przesuwanie wierzchołków, które zostały dodane do ich odpowiednich pozycji. Wymaga to jednak jeszcze większej mocy procesora, a takie metody są już wymagające procesora.

Głębokość Odrzucenia

W szczególności w tego typu grach zobaczysz tego typu rzeczy. Na przykład GTA3 używa tej techniki do zmniejszenia liczby wielokątów, które musi renderować. Pomysł polega po prostu na tym, aby nie renderować obiektów znajdujących się w pewnej odległości od kamery. To takie proste. Z pewnością zauważyłeś w grach wyścigowych, że budynki wydają się pojawiać na horyzoncie, co jest najlepszym wskaźnikiem, że ta technika była używana. Często programista użyje czegoś takiego jak mgła, aby zamaskować efekt poppingu, który powoduje pojawienie się nowego obiektu na scenie. Inną alternatywą, jaką widziałem w przeszłości, jest stopniowe zmniejszanie przezroczystości obiektu w miarę zbliżania się do kamery. Ta i mgła wykorzystują prawie tę samą ideę, która polega na zmniejszeniu gwałtowności wizualnego przepływu. Zawsze jest ładniej, gdy wszystko wydaje się płynne i stopniowe.