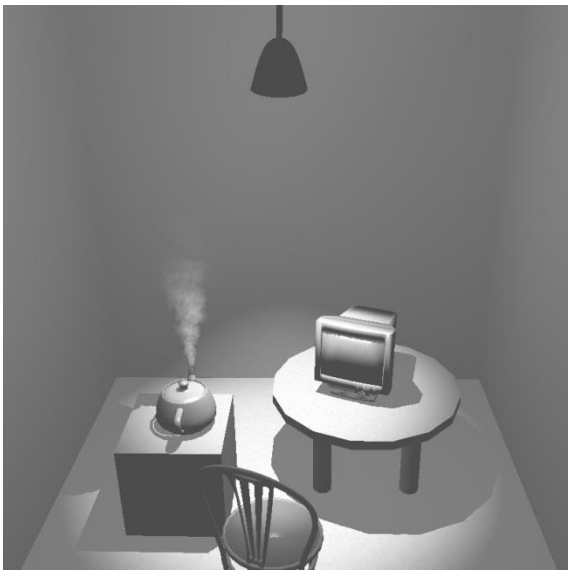
Zadanie 4: Mapy światła i cienia

Zadanie

W tym zadaniu dany jest projekt renderujący scenę przedstawiającą pokój z bujającą się lampą zawieszoną pod sufitem. Pierwszym ćwiczeniem będzie rzutowanie perspektywiczne tekstury światła (białe koło z rozmytym brzegiem na czarnym tle) na całą scenę, zgodnie z aktualnym położeniem lampy. Drugim, będzie dodanie cieni przy pomocy techniki mapy cienia. Wykorzystanie mapy cienia w scenie przebiega analogicznie do wykorzystania mapy światła, dlatego w tej części głównym problemem będzie odpowiednie zainicjalizowania parametrów tekstury oraz wyrenderowania jej zawartości – tj. bufor głębokości sceny widzianej z pozycji lampy.

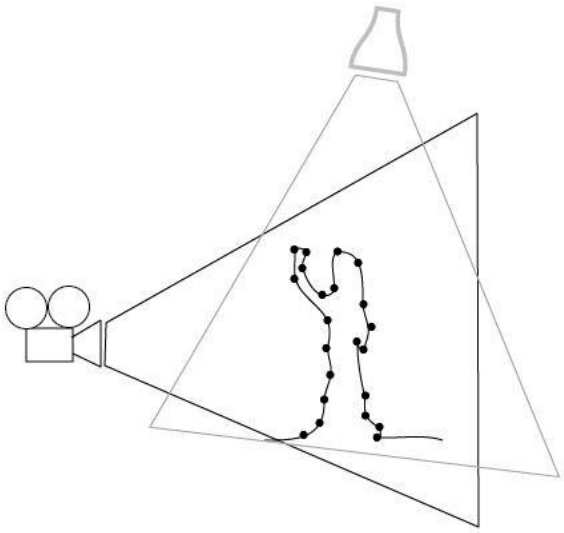
Bez map światła i cienia



Z mapami światła i cienia.

Uzupełnić należy funkcje klasy RoomDemo w pliku roomDemo.cpp oraz kod shadera pikseli w pliku lightAndShadowPS.hlsl.

1. Wyznaczanie macierzy współrzędnych tekstury
(konstruktor i funkcja UpdateLamp() klasy RoomDemo)



Perspektywiczne rzutowanie tekstury bardzo przypomina realizowanie rzutowania wierzchołków na ekran. Musimy wyznaczyć macierz przekształcenia do układu kamery (w tym wypadku lampy) i macierz rzutowania perspektywicznego. Następnie przekształcimy punkty z układu sceny (świata) przez obie macierze. Dodatkowo w celu wyznaczenia współrzędnych tekstury musimy jeszcze przeskalować i przesunąć wynik, ponieważ zakres współrzędnych tekstury jest inny niż zakres współrzędnych okna widoku (dla okna widoku współrzędne x oraz y są z zakresu [-1, 1]; dla tekstury współrzędne te są z zakresu [0, 1] – dodatkowo w przypadku współrzędnych tekstury oś Y jest odwrócona względem współrzędnych okna widoku).

Najpierw policzyć musimy macierz widoku i perspektywy lampy. Będzie to analogiczne do wyznaczania macierzy kamery „patrzącej" tak jak lampa. Położenie takiej kamery – nazwijmy je LP – w będzie znajdować się na dole lampy – punkt [0, -0.05, 0] w układzie lokalnym kamery. Kamera będzie spoglądać na punkt – nazwijmy go LT – przesunięty w kierunku ujemnym osi Y – punkt [0, -10, 0], również w układzie lokalnym lampy. Ostatnią składową jest wektor „do góry" kamery. W układzie lokalnym lampy będzie on skierowany wzdłuż osi X. Aby wyznaczyć poprawną macierz kamery musimy te dwa punkty i wektor wyrazić w układzie sceny. Posłużyć się możemy macierzą modelu lampy, którą daną mamy jako parametr funkcji oraz funkcjami XMVector3TransformCoord i XMVector3TransformNormal. Po przekształceniu ostateczną macierz wyznaczy funkcja XMMatrixLookAtLH. W późniejszym etapie potrzebna będzie nam również odwrotność tej macierzy, którą możemy od razu wyznaczyć (funkcja XMMatrixInverse).

Drugą potrzebną macierzą jest macierz rzutowania perspektywicznego dla światła. Do jej policzenia użyć należy funkcji XMMatrixPerspectiveFovLH. Szerokość kątową (w radianach) snopu światła definiuje stała LIGHT_FOV_ANGLE, odległość bliższej i dalszej płaszczyzny obcinania stałe LIGHT_NEAR i LIGHT_FAR, natomiast snop światła będzie okrągły (a nie wydłużony w elipsę wzdłuż osi X lub Y) dlatego współczynnik proporcji okna „widoku" światła wynosi 1.

Teraz ustalić możemy przekształcenie dla współrzędnych tekstury z układu sceny do układu lampy. Będzie to złożenie kolejno przekształceń: macierzy widoku lampy, macierzy perspektywy dla lampy,

macierzy skalowania XMMatrixScaling(0.5f, -0.5f, 1.0f) oraz macierzy przesunięcia XMMatrixTranslation(0.5f, 0.5f, 0.0f). Te dwie ostatnie służą przekształceniu współrzędnych okna widoku do współrzędnych tekstury.

Ostatnim elementem będzie wyznaczenie współrzędnych punkt LP (położenie światła) w układzie sceny. Program można teraz uruchomić. Na razie nie zobaczymy żadnego efektu wynikającego z policzonych przez nas macierzy, jedynie to, że położenie źródła światła na scenie śledzi teraz ruchy lampy.

2. Renderowanie z użyciem mapy światła

(konstruktor klasy RoomDemo oraz shader pikseli lightAndShadowPS.hlsl)

Tekstura mapy światła jest już wczytana i będzie podpięta do potoku renderowania razem z pozostałymi elementami naszego efektu. Musimy jednak się upewnić, że odczytywanie będzie przebiegać w odpowiedni sposób. W przypadku, gdy współrzędne tekstury wykroczą poza zakres [0, 1] chcemy by zwrócony został kolor z wartościami 0 we wszystkich czterech kanałach (w tym kanale alfa, o czym za chwilę). Ponadto użyć chcemy filtrowania biliniowego (tak by wybierany był najbliższy poziom mip-mapy, zamiast interpolacji pomiędzy sąsiednimi). Dlatego stworzyć musimy Sampler, dla którego zawijanie adresów ustawione jest na D3D11_TEXTURE_ADDRESS_BORDER, kolor ramki ustawiony jest na kolor [0, 0, 0, 0], a filtrowanie na D3D11_FILTER_MIN_MAG_LINEAR_MIP_POINT. Sampler stworzony powinien być w zaznaczonym miejscu konstruktora klasy RoomDemo.

Drugim krokiem jest wyznaczenie w shaderze pikseli współrzędnych tekstury dla danego piksela. Otrzymujemy je poprzez przekształcenie współrzędnych piksela w układzie świata (pole worldPos w strukturze PSInput) przez opisaną powyżej macierz (dostępną w zmiennej mapMtx). Aby poprawnie wykonać dzielenie perspektywiczne należy jeszcze trzy pierwsze współrzędne wynikowego wektora (*x*, *y* oraz *z*) podzielić przez wartość czwartej współrzędnej *w* (Uwaga: Warto zauważyć, że w shaderze wierzchołków po przekształceniu pozycji wierzchołka przez macierz perspektywy, nie wykonujemy dzielenia współrzędnych otrzymanego wyniku przez współrzędną *w*. Tam nie musimy tego robić, gdyż operację tą automatycznie wykonuje za nas karta graficzna w fazie rasteryzacji. W przypadku perspektywicznego rzutowania tekstury nie mamy jednak tego mechanizmu i dzielenie to musimy wykonać ręcznie). Mimo, że do adresowania tekstury używać będziemy współrzędnych *x* i *y* otrzymanego wektora, należy również wykonać dzielenie dla współrzędnej z – przyda nam się ona później przy mapie cieni.

Kolor światła używany dalej przy cieniowaniu piksela należy pobrać z mapy światła. Jest ona już zainicjalizowana i dostępna w zmiennej lightMap. Do adresowania tej tekstury należy użyć współrzędnych *x* i *y* otrzymanego w poprzednim kroku wektora. Chcemy jednak, aby piksele dla których wartość zwrócona z mapy światła jest za niska, były oświetlone pewnym domyślnym przyciemnionym światłem (jego kolor zapisany jest w zmiennej defaultLightColor). Należy więc porównać wartość zwróconą z mapy światła z wartością domyślną i wziąć większą z nich. Mapa cienia zawiera tylko odcienie szarości, porównywać można tylko jeden ze kanałów obu kolorów.

Aby zobaczyć wynik renderowania z użyciem mapy światła należy w funkcji Render w klasie Room w zaznaczonym miejscu przypiąć tekstury m_lightMap i (później nam potrzebną) m_shadowMap do etapu pixel shadera (metoda SetTextures) i narysować scenę z użyciem nowego zmodyfikowanego właśnie shadera pikseli.

3. Projekcja wsteczna

(poprawka do shadera pikseli)

Istnieje pewna różnica pomiędzy rzutowaniem geometrii na ekran i rzutowaniem w celu wygenerowania współrzędnych tekstury. W pierwszym przypadku fragmenty powierzchni nie znajdujące się pomiędzy bliższą i dalszą płaszczyzną obcinania jest wykluczana, w drugim nie. Dlatego mapa światła na powyższym obrazie rzutowana jest zarówno poniżej jak i powyżej lampy (jaśniejsze koło na suficie).

Najbardziej uniwersalnym rozwiązaniem byłoby przekazanie do shadera pikseli położenia lampy i wektora kierunku w jakim rzuca światło oraz odległości bliższej płaszczyzny obcinania wzdłuż tego wektora (płaszczyzna obcinania byłaby do tego wektora prostopadła). Dla każdego piksela należałoby sprawdzić, czy znajduje się on przed czy za tą płaszczyzną. Na szczęście odchylenia lampy są na tyle małe, że można ten przypadek trochę uprościć i zastosować płaszczyznę obcinania prostopadłą do osi OY przechodzącą przez aktualne położenie światła. Należy więc porównać położenie piksela w układzie sceny (pole worldPos struktury PSInput) z położeniem światła (zmienna lightPos) i jeżeli współrzędna y położenia piksela jest większa niż współrzędna y położenia światła, należy zastosować domyślny, przyciemniony kolor światła zamiast tego pobranego z mapy światła.

4. Inicjalizacja tekstury mapy cienia

(konstruktor klasy RoomDemo)

Zanim zaczniemy renderować zawartość mapy cieni, musimy odpowiednio zainicjalizować tą teksturę. Tekstura będzie spełniać dwa zadania: będzie służyła jako bufor głębokości przy renderowaniu sceny z punktu widzenia światła, oraz jako tekstura przy właściwym renderowaniu sceny na ekran. Dlatego będziemy musieli stworzyć dla niej dwa widoki: ID3D11DepthStencilView dla pierwszego zadania, oraz ID3D11ShaderResourceView dla drugiego.

Do poprawnej inicjalizacji samej tekstury musimy podać jej wymiary (pola Width i Height struktury Texture2DDescription – ich wartość powinna być równa stałej TEXTURE_SIZE), format (pole Format – wartość powinna być równa DXGI_FORMAT_R32_TYPELESS; musimy użyć formatu R32_TYPELESS, gdyż w używając tekstury jako bufora głębokości chcemy interpretować jej wartości jako typ

D32_FLOAT, natomiast w przypadku użycia jej jako tekstury w shaderze pikseli interpretujemy dane z niej pobierane jako R32_FLOAT. Te dwa ostatnie formaty nie są ze sobą do końca kompatybilne – mimo, że reprezentacja bitowa jest taka sama – i jedynym formatem, który można konwertować na oba z nich jest R32_TYPELESS). Pole BindFlags powinno być równe bitowej operacji OR wartości D3D11_BIND_DEPTH_STENCIL oraz D3D11_BIND_SHADER_RESOURCE, natomiast pole MipLevels powinno mieć wartość 1 (używamy tylko jednego poziomu mipmap).

Do inicjalizacji obiektu ID3D11DepthStencilView dla mapy cieni wystarczy zmodyfikować format (pole Format struktury DepthStencilViewDescription, którego wartość powinna być równa DXGI_FORMAT_D32_FLOAT).

Struktura ShaderResourceViewDescription opisująca obiekt ID3D11ShaderResourceView dla mapy cieni powinna w polu Format mieć wartość DXGI_FORMAT_R32_FLOAT, pole ViewDimensions wartość D3D11_SRV_DIMENSION_TEXTURE2D (wskazując, że tekstura jest dwuwymiarowa), pole Texture2D.MipLevels wartość 1, a pole Texture2D.MostDetailedMip wartość 0.

5. Renderowanie do mapy cienia

(funkcja BeginShadowRender() w klasie LightAndShadowMap oraz Render() klasy RoomDemo)

Przekształcenie perspektywiczne (takie jakie występuje przy rysowaniu na ekranie, czy też to, które wykorzystujemy przy wyliczaniu współrzędnych tekstury w shaderze pikseli) powoduje (po znormalizowaniu, czyli podzieleniu wyniku przez współrzędną *w*), że dla punktów pomiędzy bliższą i dalszą płaszczyzną obcinania, wartości współrzędnej z zostają nieliniowo „ściśnięta” do przedziału [0,1]. Dla każdego kierunku w polu widzenia lampy możemy wyznaczyć taką wartość dla najbliższego lampie w danym kierunku punktu sceny po prostu przez wyrenderowanie wszystkich obiektów z punktu widzenia lampy. Szukane przez nas wartości automatycznie trafią do bufora głębokości (tak działa algorytm z-bufora), którego później użyjemy jako mapy cienia. W kolejnym punkcie przy renderowaniu piksela będziemy mogli pobrać wartość *z* w kierunku od lampy do piksela, porównać z wartością tex.z, którą już wyznaczamy i dzięki temu określić, czy punkt jest najbliższym lampie w danym kierunku czy nie.

W celu wypełnienia mapy cienia musimy przygotować renderowanie z punktu widzenia lampy. Musimy więc:

- Skopiować wyznaczone przez nas wcześniej macierze widoku (wraz z jej odwrotnością) i perspektywy lampy do odpowiednich buforów – metoda UpdateBuffe()r.
- Ustawić viewport rozmiaru tekstury (TEXTURE_SIZE) – struktura ViewportDescription i metoda RSSetViewports kontekstu renderowania
- Podpiąć naszą teksturę cieni (a właściwie jej DephtStencilView) jako bufor głębokości – metoda OMSetRenderTargets kontekstu renderowania (tutaj renderowanie do bufora koloru nie jest nam do niczego potrzebne, dlatego należy tablicę render target-ów zostawić pustą, tj. przekazać 0 w dwóch pierwszych parametrach)
- Wyczyścić bufor głębokości z danych z poprzedniej klatki – metoda ClearDephtStencilView kontekstu renderowania

Sam proces renderowania modeli oraz cząsteczkek należy przeprowadzić z użyciem bazowego efektu Phonga.

6. Renderowanie z użyciem mapy cienia.

(shader pikseli lightAndShadowPS.hlsl, drobne poprawki w UpdateLamp klasy RoomDemo)

Do adresowania mapy cieni w shaderze pikseli używamy tych samych współrzędnych, co do adresowania mapy światła. Różnica będzie tylko w wartości zwracanej z metody Sample. W przypadku mapy światła był to kolor (typ float4), natomiast samplowanie mapy cieni zwróci wartość tylko w jednym kanale (czerwonym). Wartość ta jest wartością z bufora głębokości w miejscu, gdzie znalazłby się aktualnie rozpatrywany piksel przy renderowaniu sceny z punktu widzenia lampy. Wartość tą należy porównać z wartością współrzędnej z wektora wyznaczanego w punkcie 2. Jeżeli wartość pobrana z mapy cienia jest mniejsza niż wartość współrzędnej z tego wektora, to znaczy, że na drodze pomiędzy tym pikselem a źródłem światła, przy renderowaniu sceny z punku widzenia lampy znalazł się jakiś inny obiekt, czyli aktualnie rozpatrywany piksel znajduje się w cieniu. Dla takiego piksela, zamiast koloru światła pobranego z mapy światła, należy użyć koloru domyślnego (defaultLightColor).

Po uruchomieniu programu powinny pojawić się znaczne artefakty. Wynika to z faktu, że obliczanie współrzędnej z zrzutowanej perspektywicznie z punktu widzenia lampy nie są dokładne – nie liczymy najbliższych punktów we wszystkich kierunkach, lecz tylko tych, które przechodzą przez środki pikseli bufora głębokości, a piksel na ekranie nigdy nie odpowiada bezpośrednio środkowi piksela w mapie cienia. Aby temu zapobiec należy zwróconą współrzędną *z* zmniejszyć o jakąś małą wartość. Najłatwiej to zrobić modyfikując funkcję UpdateLamp() klasy RoomDemo, gdzie tworzona jest macierz przekształcenia współrzędnych tekstury (patrz punkt 1). Częścią składową tej macierzy jest macierz translacji – należy ją zmodyfikować tak, by dodatkowo następowało przesunięcie wzdłuż osi Z o wartość -0.00001f.

7. Uwagi końcowe

Mapy cienia są jedną z możliwych metod uzyskania cieni dla dynamicznej sceny. Inną popularną metodą są bryły cienia. Dla brył cienia uzyskuje się dużą dokładność wyznaczenia krawędzi cienia, podczas gdy dla map cienia może pojawić się aliasing jego krawędzi (jego wielkość zależy tutaj od rozdzielczości mapy cienia oraz kąta między kierunkiem światła i kierunkiem patrzenia) i błędy określenia czy obiekt jest w cieniu dla płaszczyzn prawie równoległych do kierunku światła. Z kolei mapy cienia można wykorzystać w połączeniu z teksturami zawierającymi maskę przezroczystości (np. prostokąty reprezentujące liście na drzewie z nałożoną teksturą posiadającą odpowiednią maskę przezroczystości), zaś bryły cienia operują wyłącznie na krawędziach geometrii sceny. Istnieje wiele odmian map cienia, które poprawiają ich precyzję (Trapezoidal Shadow Maps, Perspective Shadow Maps) oraz zapewniają lepsze filtrowanie i rozmycie(Variational Shadow Maps). Osobnym problemem, ale również dość dobrze już opracowanym, jest liczenie realistycznych cieni dla świateł powierzchniowych oraz światła otoczenia (Ambient Occlusion).