Uniwersytet Mikołaja Kopernika
Wydział Fizyki, Astronomii i Informatyki Stosowanej
Zakład Mechaniki Kwantowej

Radosław Płoszajczak
nr albumu: 177.823

Praca magisterska na kierunku
fizyka techniczna

Techniki generowania cieni w grafice 3D z użyciem OpenGL

Opiekun pracy dyplomowej
dr Jacek Matulewski
Zakład Mechaniki Kwantowej

Toruń 2008

Pracę przyjmuję i akceptuję

.....
data i podpis opiekuna pracy

Potwierdzam złożenie pracy dyplomowej

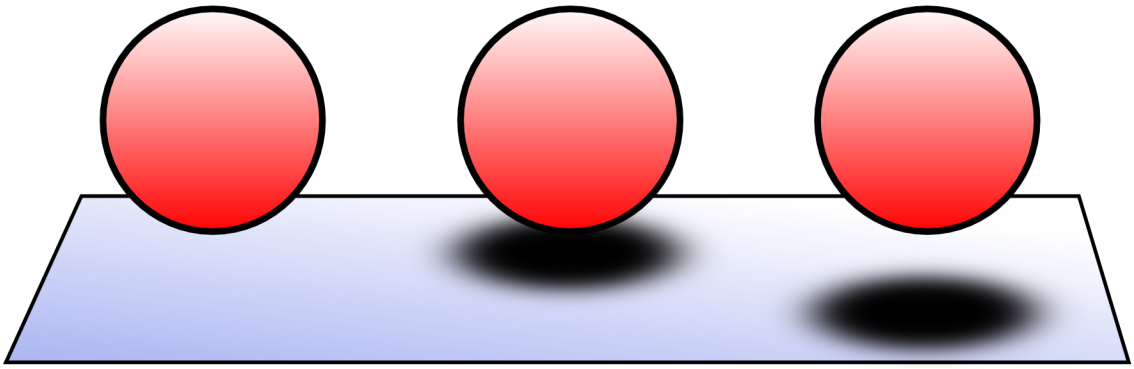
.....
data i podpis pracownika dziekanatu

Dziękuję doktorowi Jackowi Matulewskiemu za pomoc podczas tworzenia niniejszej pracy oraz mojej Rodzinie za wsparcie, na które zawsze mogłem liczyć.

Spis treści

1. Wstęp

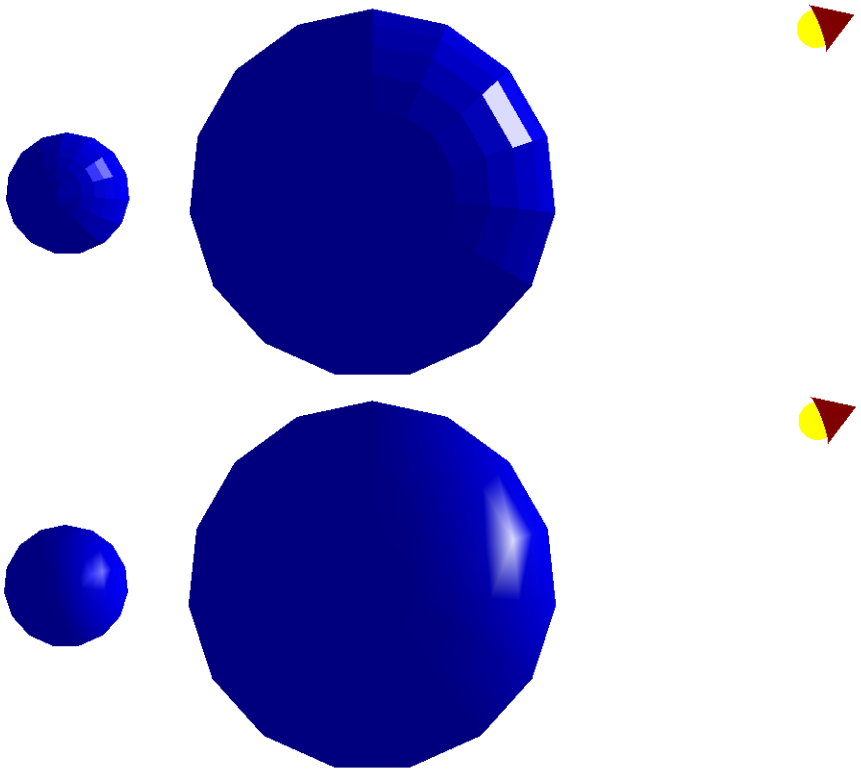
Istotnym elementem widzianego przez człowieka świata są cienie. Pozwalają one lepiej postrzegać odległości i wzajemne położenie obiektów. Uwydatniają fakturę powierzchni wzbogacając powierzchnie o nowe detale. Są również inspiracją dla sztuki, a nawet ją tworzą czego doskonałym przykładem jest chiński teatr cieni oraz hiszpański teatr za ekranem. W kinematografii cienie stały się podstawowym narzędziem budowania atmosfery wielu filmowanych scen. Dla fotografa zabawy z cieniem dają prawie nieograniczone możliwości wykonania interesujących i zaskakujących ujęć.



Rys. 1. Rola cienia w postrzeganiu odległości.

Z definicji [1] cień jest to „obszar, do którego światło nie dochodzi zupełnie (cień pełny) lub dochodzi częściowo (półcień). Większość popularnych interfejsów programistycznych do tworzenia grafiki 3D w czasie rzeczywistym, takich jak OpenGL czy Direct3D, udostępnia model oświetlenia, który tylko częściowo realizuje to zjawisko. Uwzględnia on tylko cienie własne tj. zmianę jasności oświetlonej powierzchni w zależności od jej ustawienia względem źródła światła i obserwatora. Pomija natomiast możliwość blokowania promieni przez poszczególne obiekty sceny (cienie rzucane przez obiekty na inne obiekty). Do obliczania jasności poszczególnych fragmentów powierzchni używany jest model Phonga z wykorzystaniem cieniowania płaskiego lub cieniowania Guarda do interpolacji kolorów pomiędzy wierzchołkami (zob. rys. 2). Takie rozwiązanie zostało zastosowane głównie ze względu na małą złożoność obliczeniową, przy stosunkowo realistycznym efekcie. W przypadku grafiki 3D czasu rzeczywistego, do której służą wymienione wyżej biblioteki, uwzględnienie zjawiska przesłaniania światła w sposób ogólny mogłoby być bardzo nieefektywne w konkretnych zastosowaniach. Z tego właśnie powodu

programista skazany jest na własną implementację, przy czym musi wybrać najlepszy do swoich zastosowań algorytm.



Rys. 2. Model oświetlenia Phonga dostępny w OpenGL. Górny obrazek przedstawia cieniowanie płaskie, natomiast dolny cieniowanie gładkie (Guarda).

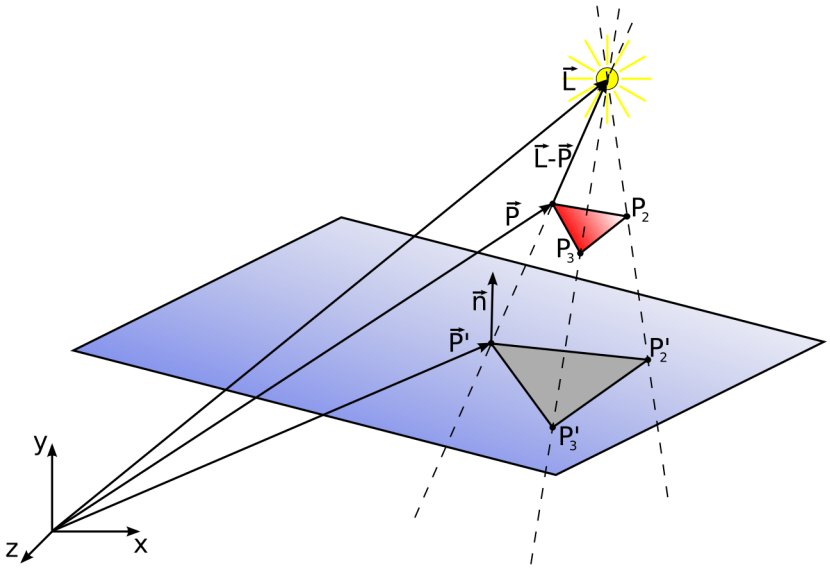
W dalszej części pracy zostaną opisane trzy podstawowe techniki generowania cieni odpowiednie dla wyświetlania grafiki w czasie rzeczywistym. Pierwsza najprostsza metoda rzutowania cieni (ang. *projected shadows*) opiera się na wykorzystaniu specjalnie przygotowanej macierzy przekształcenia służącej do wykonania „spłaszczenia" obiektu tj. zrzutowania go na płaszczyznę, na którą rzuca on cień. Bardziej zaawansowanym algortytmem są tzw. cienie objętościowe (ang. *volumetric shadows*). W tym algorytmie buduje się bryłę w tej części przestrzeni, gdzie dostęp światła jest ograniczony, co w połączeniu z odpowiednim rysowaniem daje w rezultacie bardzo realistyczne cienie na złożonych obiektach. Ostatnia z opisywanych technik jest najczęściej stosowana we współczesnych programach i polega na zastosowaniu mapy głębokości, która następnie jest rzutowana na geometrię jak z projektora, co ostatecznie pozwala określić zacienione obszary. Metoda ta nazywana jest mapowaniem cieni (ang. *shadow mapping*).

2. Rzutowanie cieni (projected shadows)

Książki opisujące podstawy programowania grafiki trójwymiarowej rzadko traktują o technikach generowania cieni. Jeżeli ten temat jest już poruszany najczęściej przetaczaną metodą jest rzutowanie geometrii (np. [2]). Dzieje się tak głównie ze względu na jej prostotę i łatwość implementacji. Ponadto w przypadku małych programów technika ta daje zadowalające efekty przy minimalnym nakładzie pracy.

2.1. Zasada działania i metody rysowania

Analizując zagadnienie z punktu widzenia geometrii cień jest rzutem obiektu przysłaniającego światło na obiekt zasłaniany. W przypadku punktowego źródła światła położonego w stosunkowo niedużej odległości od obiektu mamy do czynienia z rzutem perspektywicznym (rys.3). Oddalając jego pozycję aż do nieskończoności (np. w przypadku światła słonecznego) promienie stają się równoległe, a więc rzut staje się równoległy. Oba przypadki można przedstawić w jednolity sposób we współrzędnych jednorodnych (o tym niżej, zob. również dodatek A).

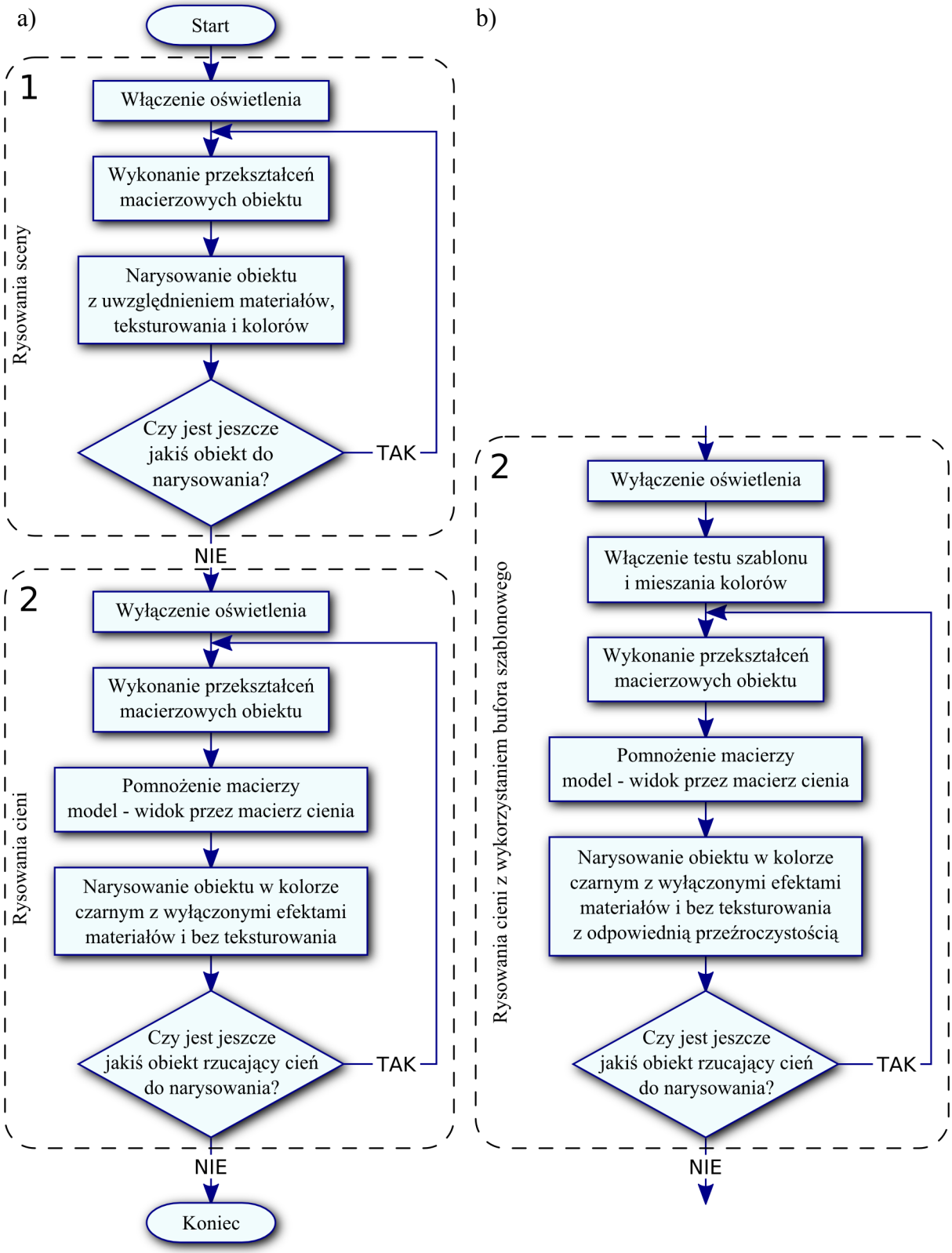


Rys. 3. Tworzenie cienia poprzez rzutowanie perspektywiczne obiektu na płaszczyznę.

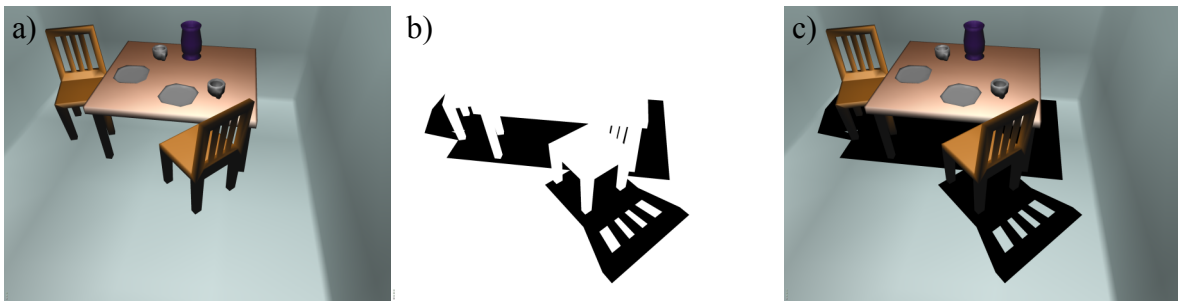
Idea metody rzutowania opiera się na wykorzystaniu macierzy przekształcenia

dokonującej rzutowania perspektywicznego obiektu na płaszczyznę przyjmującą cień. Dzięki temu w prosty sposób można wykorzystać te same współrzędne wierzchołków modelu (we współrzędnych modelu) do jego rysowania jak również do wygenerowania jego cienia. Biblioteka OpenGL pozwala załadować macierz model-widok (patrz dodatek C) służącą do przekształcenia obiektu w przestrzeni sceny lub pomnożyć aktualną macierz przez nową co skutkuje tym, że każdy rysowany od tego momentu wierzchołek podlega odpowiedniemu złożeniu transformacji.

Sposób rysowania sceny wykorzystujący standardowy model oświetlenia OpenGL wzbogacony o cienie rzucane przez obiekty przedstawia zamieszczony poniżej schemat blokowy (rys. 4a). Cały proces tworzenia obrazu można podzielić na dwa etapy. Pierwszy z nich jest zwykłym renderingiem (rys. 5a), drugi zaś ma za zadanie wygenerowanie cieni (rys. 5b) z jednoczesnym nałożeniem ich na obraz z etapu pierwszego (rys. 5c). W pierwszym etapie rysowania należy włączyć efekty oświetlenia. Ponadto należy mieć na uwadze, że każdy obiekt przed renderowaniem może podlegać odpowiednim przekształceniom obracania, przesuwania, skalowania i innych zdefiniowanych przy pomocy odpowiedniej macierzy dlatego w następnym kroku należy ją załadować do macierzy model-widok. Gdy transformacje są już wprowadzone można przystąpić do rysowania prymitywów składających się na dany obiekt. W pierwszym przebiegu mogą one mieć zdefiniowane stosowne kolory, materiały oraz tekstury. Powyższe czynności należy powtórzyć dla każdego obiektu sceny. Rezultatem pierwszego etapu jest obraz wykonany z wykorzystaniem wszystkiego tego, co standardowo oferuje biblioteka OpenGL (rys. 5a). Drugi przebieg renderingu ma na celu udoskonalenie go poprzez zacienienie odpowiednich obszarów. Przed przystąpieniem do rysowania w drugim etapie należy wyłączyć oświetlenie ponieważ jego użycie byłoby nadmiarowe. Cienie będą czarnymi obszarami zbudowanymi z tych samych wielokątów co rzucające je obiekty dlatego należy pozbawić je kolorów, właściwości materiałów oraz tekstur. Rendering kolejnych obiektów przebiega podobnie jak w etapie pierwszym z tą jednak różnicą, że po dokonaniu transformacji każdego z obiektów powstała w ten sposób macierz model-widok musi zostać jeszcze pomnożona przez macierz rzutowania cienia.

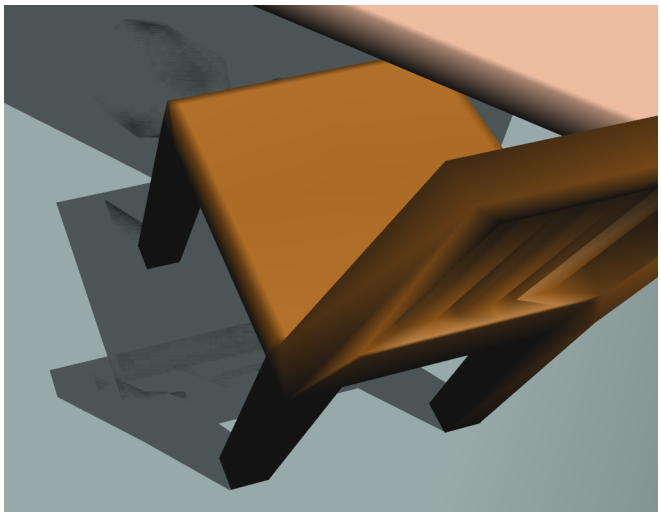


Rys. 4. a) Schemat blokowy rysowania sceny z cieniami, **b)** modyfikacja przebiegu renderowania cieni wykorzystująca bufor szablonowy i mieszanie kolorów.



Rys. 5. Poszczególne etapy tworzenia obrazu oraz wynikowy rendering. Kolejno od lewej: **a)** pierwszy etap rysowania (oświetlona scena), **b)** cienie utworzone w drugim etapie rysowania, **c)** ostateczny obraz powstały w drugim przebiegu renderingu.

Opisany powyżej dwuprzebiegowy algorytm rysowania pozwala w rezultacie otrzymać realistycznie wyglądającą scenę z czarnymi obszarami wyznaczającymi miejsca, do których światło nie dociera. Jest to niestety wyidealizowany przypadek, który nie bierze pod uwagę wpływu światła rozproszonego (półcieni). Gdyby je uwzględnić okazuje się że cienie nie będą już tylko czarnymi plamami, a obszarami odpowiednio przyciemnionymi. Efekt ten można uzyskać wykorzystując mechanizm mieszania kolorów dostępny w OpenGL. Polega on na użyciu kanału alfa koloru, który w tym przypadku będzie określał przezroczystość prymitywów składających się na cienie. Zastosowanie tego narzędzia nie jest jednak wystarczające ponieważ jak już wcześniej zostało pokazane rysowane są spłaszczone obiekty czego efektem ubocznym jest wzajemne nakładanie się tworzących je trójkątów. Dla w pełni czarnych cieni nie ma to znaczenia, ale w przypadku półcieni rysowanie przeźroczystych powierzchni skutkuje wielokrotnym przyciemnieniem danego obszaru (zob. rys. 6).



Rys. 6. Błędy podczas rysowania cienia krzesła. Spłaszczone obiektu powoduje, że nakładają się na siebie elementy oparcia oraz siedziska czego rezultatem jest wielokrotne przyciemnienie fragmentu podłogi.

Aby zapobiec takim artefaktom należy wykorzystać bufor szablonowy (ang. *stencil buffer*, patrz dodatek D). Posłuży on do zapamiętania pikseli obrazu, które zostały już

zaciemnione. Modyfikacja poprzedniej metody ograniczy się do zmiany drugiego przebiegu renderingu (rys. 4b). Po wyłączeniu oświetlenia należy uruchomić mechanizmy mieszania kolorów oraz testu szablonowego. Pierwszy z nich nie wymaga specjalnej konfiguracji i może wykorzystywać domyślne zachowanie biblioteki OpenGL. Drugi zaś musi zostać precyzyjne określony. Metoda zakłada, że na początku bufor szablonu dla każdego piksela ma przypisaną wartość `0`. Jeżeli punkt obrazu zostanie zaciemniony jego wartość zmieniona zostaje na `1`. Ponadto test szablonu mają przechodzić tylko te fragmenty renderingu, dla których wartość ta wynosi `0` co w rezultacie zapobiega ich ponownemu zaciemnianiu.

2.2. Wyprowadzenie macierzy przekształcenia

W poniższych obliczeniach przydatnych będzie kilka wzorów oraz znajomość podstawowych własności współrzędnych jednorodnych (patrz dodatek A). Równanie kierunkowe prostej przechodzącej przez 2 punkty o wektorach wodzących $\vec{P}$ i $\vec{L}$ wyraża się następująco:

$$\vec{x} = \vec{P} + \lambda(\vec{L} - \vec{P}) . \qquad (1)$$

Parametr λ określa położenie na tej prostej i przyjmuje wartość równą 0 dla $\vec{P}$ oraz 1 dla $\vec{L}$. Punkt $\vec{L}$ można identyfikować ze źródłem światła, a $\vec{P}$ z położeniem wierzchołka, którego rzut obliczamy. Płaszczyznę we współrzędnych jednorodnych [3] opisuje następujące równanie:

$$A x + B y + C z + D w = 0 , \qquad (2)$$

w którym współczynniki A, B, C są współrzędnymi wektora normalnego do płaszczyzny, a D jest odległością płaszczyzny od środka układu współrzędnych wziętą ze znakiem minus. W poniższym wyprowadzeniu płaszczyzna ta jest rzutnią, to jest płaszczyzną, na której szukamy punktu $\vec{P'}$ będącego rzutem punktu $\vec{P}$ wzdłuż osi PL.

Zgodnie z rysunkiem 3 cień obiektu wyznaczają rzuty punktów $\vec{P}$ obiektu na powierzchnię, na którą on pada (rzutnia). Projekcję danego wierzchołka określa miejsce przecięcia prostej przechodzącej przez źródło światła $\vec{L}$ i rzutowany punkt z płaszczyzną rzutni (2). W pierwszej kolejności należy wyznaczyć współrzędne dowolnego punktu na prostej PL wyrażone poprzez wartość parametru λ. Robimy to podstawiając współrzędne wektorów wodzących $\vec{P}$ i $\vec{L}$ do równania (1):

$$\vec{x}=(P_x,P_y,P_z,P_w)+\lambda((L_x,L_y,L_z,L_w)-(P_x,P_y,P_z,P_w))=$$

$$(P_x,P_y,P_z,P_w)+\lambda((L_xP_w-P_xL_w,L_yP_w-P_yL_w,L_zP_w-P_zL_w,L_wP_w))= \quad (3)$$

$$(P_xL_w+\lambda(L_xP_w-P_xL_w),P_yL_w+\lambda(L_yP_w-P_yL_w),P_zL_w+\lambda(L_zP_w-P_zL_w),L_wP_w)$$

Kolejną czynnością jest podstawienie powyższego wyniku (3) do związku opisującego rzutnię (2):

$$\begin{aligned}&A[P_xL_w+\lambda(L_xP_w-P_xL_w)]+B[P_yL_w+\lambda(L_yP_w-P_yL_w)]\\&+C[P_zL_w+\lambda(L_zP_w-P_zL_w)]+D(L_wP_w)=0\end{aligned} \quad (4)$$

Po przekształceniach można wyznaczyć konkretną wartość parametru λ określając punkt na prostej, w którym przecina ona rzutnię:

$$\lambda=-\frac{L_w(AP_x+BP_y+CP_z+DP_w)}{A(L_xP_w-P_xL_w)+B(L_yP_w-P_yL_w)+C(L_zP_w-P_zL_w)}. \quad (5)$$

Wstawienie otrzymanej wartości parametru λ z powrotem do równania prostej PL (1) pozwala obliczyć współrzędne punktu $\vec{P'}$ tj. rzutu punktu $\vec{P}$ na płaszczyznę:

$$\begin{aligned}\vec{P'}=(P_x,P_y,P_z,P_w)&-\frac{L_w(AP_x+BP_y+CP_z+DP_w)}{A(L_xP_w-P_xL_w)+B(L_yP_w-P_yL_w)+C(L_zP_w-P_zL_w)}\\&\cdot[(L_x,L_y,L_z,L_w)-(P_x,P_y,P_z,P_w)]\end{aligned} \quad (6)$$

Wykorzystując zasadę dodawania dla trójwymiarowej przestrzeni we współrzędnych jednorodnych (zob. wzór (64) w dodatku A) można obliczyć różnicę wektorów $\vec{P}$ i $\vec{L}$:

$$\begin{aligned}\vec{P'}=(P_x,P_y,P_z,P_w)&-\frac{L_w(AP_x+BP_y+CP_z+DP_w)}{A(L_xP_w-P_xL_w)+B(L_yP_w-P_yL_w)+C(L_zP_w-P_zL_w)}\\&\cdot(L_xP_w-P_xL_w,L_yP_w-P_yL_w,L_zP_w-P_zL_w,L_wP_w)\end{aligned} \quad (7)$$

Zgodnie z własnością (65) z dodatku A mianownik ze stosunku określającego λ we wzorze (5) może zostać wprowadzony do współrzędnej w wektora powstałego z powyższej różnicy (7).

$$\begin{aligned}\vec{P'}=&(P_x,P_y,P_z,P_w)-L_w(AP_x+BP_y+CP_z+DP_w)\\&\cdot(L_xP_w-P_xL_w,L_yP_w-P_yL_w,L_zP_w-P_zL_w,\\&L_wP_w(A(L_xP_w-P_xL_w)+B(L_yP_w-P_yL_w)+C(L_zP_w-P_zL_w)))\end{aligned} \quad (8)$$

Powstałe wyrażenie jest dość zawiłe dlatego warto je uprościć wprowadzając nowe zmienne s (9) i t (10):

$$s=AP_x+BP_y+CP_z+DP_w \quad (9)$$

$$t=A(L_xP_w-P_xL_w)+B(L_yP_w-P_yL_w)+C(L_zP_w-P_zL_w) \quad (10)$$

Użycie powyższych zmiennych oraz przeskalowanie prawego wektora w różnicy (8) przez s pozwala zapisać ten związek w prostszej postaci:

$$\begin{aligned}\overrightarrow{P'} = &(P_x, P_y, P_z, P_w) \\ &-(L_w s(L_x P_w - P_x L_w), L_w s(L_y P_w - P_y L_w), L_w s(L_z P_w - P_z L_w), L_w P_w t)\end{aligned} \quad (11)$$

Wykonanie ostatniego odejmowania wektorów prowadzi do określenia współrzędnych rzutu punktu $\vec{P}$ na rzutnię tj. punktu $\overrightarrow{P'}$:

$$\begin{aligned}\overrightarrow{P'} = &(P_x L_w t - L_w s(L_x P_w - P_x L_w), P_y L_w t - L_w s(L_y P_w - P_y L_w), \\ &P_z L_w t - L_w s(L_z P_w - P_z L_w), L_w P_w t)\end{aligned} \quad (12)$$

Biorąc pod uwagę właściwość (63) z dodatku A można skrócić wyrażenie poprzez częściowe pozbycie się parametru L_w:

$$\overrightarrow{P'} = (P_x t - s(L_x P_w - P_x L_w), P_y t - s(L_y P_w - P_y L_w), P_z t - s(L_z P_w - P_z L_w), P_w t) \quad (13)$$

Wektor w tej postaci można rozpisać na poszczególne współrzędne, których równania po uzupełnieniu o zależności kryjące się pod oznaczeniami s (9) i t (10) oraz wymnożeniu nawiasów przyjmują formy przedstawione w równaniach (14), (15), (16) i (17).

$$\begin{aligned}P_x' = &P_x A L_x P_w - P_x A P_x L_w + P_x B L_y P_w - P_x B P_y L_w - P_x C L_z P_w - P_x C P_z L_w \\ &- P_x A L_x P_w - P_y B L_x P_w - P_z C L_x P_w - P_w D L_x P_w + P_x A P_x L_w + P_y B P_x L_w \\ &+ P_z C P_x L_w + P_w D P_x L_w\end{aligned} \quad (14)$$

$$\begin{aligned}P_y' = &P_y A L_x P_w - P_y A P_x L_w + P_y B L_y P_w - P_y B P_y L_w - P_y C L_z P_w - P_y C P_z L_w \\ &- P_x A L_y P_w - P_y B L_y P_w - P_z C L_y P_w - P_w D L_y P_w + P_x A P_y L_w + P_y B P_y L_w \\ &+ P_z C P_y L_w + P_w D P_y L_w\end{aligned} \quad (15)$$

$$\begin{aligned}P_z' = &P_z A L_x P_w - P_z A P_x L_w + P_z B L_y P_w - P_z B P_y L_w - P_z C L_z P_w - P_z C P_z L_w \\ &- P_x A L_z P_w - P_y B L_z P_w - P_z C L_z P_w - P_w D L_z P_w + P_x A P_z L_w + P_y B P_z L_w \\ &+ P_z C P_z L_w + P_w D P_z L_w\end{aligned} \quad (16)$$

$$P_w' = P_w(A L_x P_w - A P_x L_w + B L_y P_w - B P_y L_w + C L_z P_w - C P_z L_w) \quad (17)$$

Wyrażenia (14-16) na P_x', P_y' i P_z' można uprościć skracając odpowiednie wyrazy. W efekcie uzyskamy:

$$P_x' = P_x B L_y P_w + P_x C L_z P_w - P_y B L_x P_w - P_z C L_x P_w - P_w D L_x P_w + P_w D P_x L_w \quad (18)$$

$$P_y' = P_y A L_x P_w + P_y C L_z P_w - P_x A L_y P_w - P_z C L_y P_w - P_w D L_y P_w + P_w D P_y L_w \quad (19)$$

$$P_z' = P_z A L_x P_w + P_z B L_y P_w - P_x A L_z P_w - P_y B L_z P_w - P_w D L_z P_w + P_w D P_z L_w \quad (20)$$

Następnym krokiem jest pogrupowanie wyrazów w równaniach (18), (19), (20) oraz (17) przy współrzędnych P_x, P_y, P_z i P_w. Zwróćmy ponadto uwagę, że współrzędna P_w występuje we wszystkich wyrazach, można ją zatem przenieść przed nawias.

$$P_x' = P_w[P_x(B L_y + C L_z + D P_w) + P_y(-B L_x) + P_z(-C L_x) + P_w(-D L_x)] \quad (21)$$

$$P_y' = P_w[P_x(-A L_y) + P_y(A L_x + C L_z + D P_w) + P_z(-C L_y) + P_w(-D L_y)] \quad (22)$$

$$P_z' = P_w[P_x(-A L_z) + P_y(-B L_z) + P_z(A L_x + B L_y + D P_w) + P_w(-D L_z)] \quad (23)$$

$$P_w' = P_w[P_x(-A L_w)+P_y(-B L_w)+P_z(-C L_w)+P_w(A L_x+B L_y+C L_y)] \quad (24)$$

Ponowne skorzystanie z własności (63) umożliwia pozbycie się parametru P_w sprzed nawiasów kwadratowych. Tak przekształcony wektor w zapisie kolumnowym przyjmuje następującą postać:

$$\overrightarrow{P'} = \begin{bmatrix} P_x(B L_y+C L_z+D L_w)+P_y(-B L_x)+P_z(-C L_x)+P_w(-D L_x) \\ P_x(-A L_y)+P_y(A L_x+C L_z+D L_w)+P_z(-C L_y)+P_w(-D L_y) \\ P_x(-A L_z)+P_y(-B L_z)+P_z(A L_x+B L_y+D L_w)+P_w(-D L_z) \\ P_x(-A L_w)+P_y(-B L_w)+P_z(-C L_w)+P_w(A L_x+B L_y+C L_z) \end{bmatrix} \quad (25)$$

Jest to wzór pozwalający na obliczenie współrzędnych punktu $\overrightarrow{P'}$ będącego rzutem perspektywicznym punktu $\vec{P}$ na płaszczyznę przyjmującą cień tego punktu oświetloną z punktu $\vec{L}$. Przyjmując w powyższym wzorze położenie punktowego źródła światła $\vec{L}$ oraz współczynniki rzutni (A, B, C i D) jako parametry możemy łatwo znaleźć macierz M wyrażoną jedynie przez te wielkości, która pomnożona przez współrzędne punktu $\vec{P}$ da współrzędne jego rzutu $\overrightarrow{P'}$ (por. regułę przekształcenia macierzowego (68) z dodatku B):

$$\overrightarrow{P'} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \cdot \begin{bmatrix} P_x \\ P_y \\ P_z \\ P_w \end{bmatrix} = \begin{bmatrix} m_{11}\cdot P_x+m_{12}\cdot P_y+m_{13}\cdot P_z+m_{14}\cdot P_w \\ m_{21}\cdot P_x+m_{22}\cdot P_y+m_{23}\cdot P_z+m_{24}\cdot P_w \\ m_{31}\cdot P_x+m_{32}\cdot P_y+m_{33}\cdot P_z+m_{34}\cdot P_w \\ m_{41}\cdot P_x+m_{42}\cdot p_y+m_{43}\cdot P_z+m_{44}\cdot P_w \end{bmatrix} \quad (26)$$

Znając postać prawej strony powyższego równania z wyrażenia (25) możliwe jest odczytanie poszczególnych elementów macierzy przekształcenia M (26):

$$M = \begin{bmatrix} B L_y+C L_z+D L_w & -B L_x & -C L_x & -D L_x \\ -A L_y & A L_x+C L_z+D L_w & -C L_y & -D L_y \\ -A L_z & -B L_z & A L_x+B L_y+D L_w & -D L_z \\ -A L_w & -B L_w & -C L_w & A L_x+B L_y+C L_z \end{bmatrix} \quad (27)$$

Wartości na przekątnej można uprościć stosując oznaczenie:

$$d = A L_x+B L_y+C L_z+D L_w, \quad (28)$$

co daje ostateczną postać macierzy rzutowania cienia:

$$M = \begin{bmatrix} d-A L_x & -B L_x & -C L_x & -D L_x \\ -A L_y & d-B L_y & -C L_y & -D L_y \\ -A L_z & -B L_z & d-C L_z & -D L_z \\ -A L_w & -B L_w & -C L_w & d-D L_w \end{bmatrix} \quad (29)$$

Każdy z jej elementów zależy tylko od parametrów A, B, C i D równania płaszczyzny oraz wektora położenia światła $\vec{L}$. Zatem w konkretnej scenie, przy ustalonej płaszczyźnie

podłoża (rzutni) i modelu (zbioru punktów $\vec{P}$), obliczenie cienia (zbioru punktów $\vec{P'}$) oznacza wykonanie odpowiedniej liczby mnożeń współrzędnych punktów $\vec{P}$ przez stałą macierz M. W przypadku gdy obiekty sceny są animowane dla każdej kolejnej klatki animacji, gdzie zarówno położenie źródła światła, jak i położenie rzutni, mogą ulec zmianie, konieczne jest ponowne obliczenie macierzy M. Zauważmy, że macierz (29) wyrażona jest we współrzędnych jednorodnych. Oznacza to, że wektor $\vec{L}$ może być wyrażony zarówno we współrzędnych euklidesowych ze współrzędna $w=1$, jak i może znajdować się w nieskończoności. Wówczas $w=0$. Wzór ten obejmuje zatem oba przypadki rzutu wymienione na początku tego rozdziału: rzut perspektywiczny i rzut równoległy. Typ rzutu nie ma znaczącego wpływu na szybkość obliczeń.

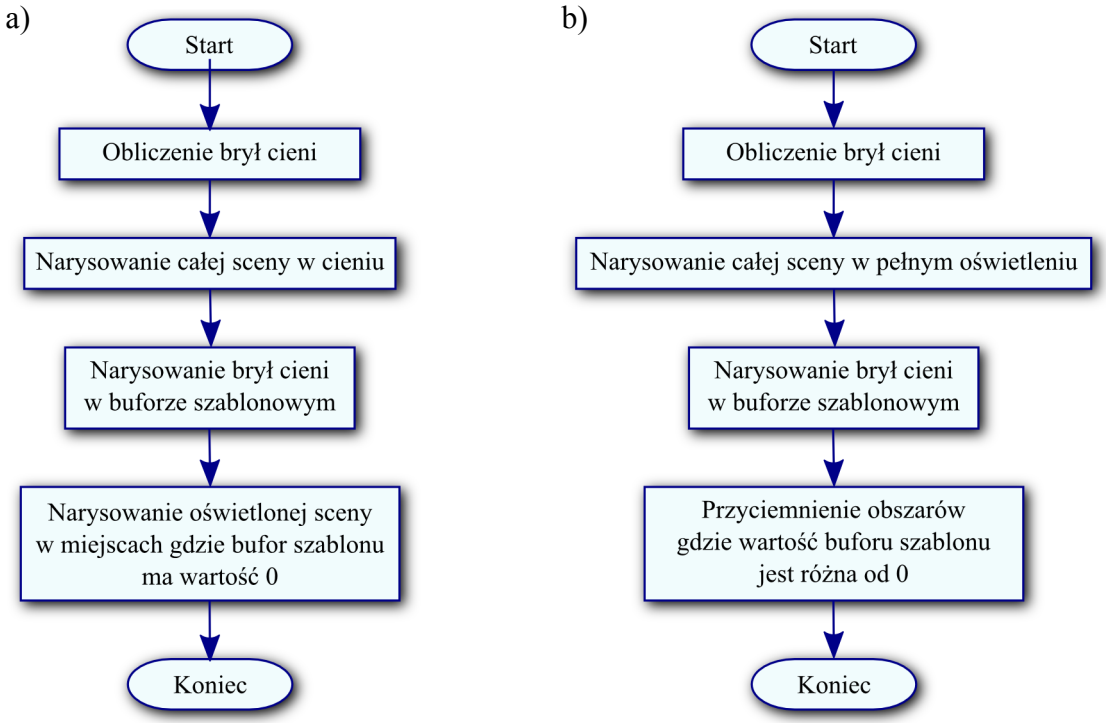
3. Cienie objętościowe (volumetric shadows)

Opisana w poprzednim rozdziale metoda rzutowania geometrii daje zadowalające efekty przy jednocześnie stosunkowo niskim nakładzie obliczeniowym. Ma ona jednak poważną wadę jakim jest ograniczenie zakresu jej działania do pojedynczej płaszczyzny. Zastosowanie w aplikacji metody cieni objętościowych (ang. *volumetric shadows*) rozwiązuje powyższy problem. Podobnie jak w przypadku algorytmu z poprzedniego rozdziału cienie rysowane są w postaci wielokątów (złożonych z trójkątów) tak więc można go również zaliczyć do algorytmów bazujących na przekształcaniu geometrii. Cienie objętościowe mają jednak zasadniczą przewagę nad rzutowaniem geometrii gdyż mogą być w poprawny sposób odwzorowywane na dowolnie zakrzywionych powierzchniach. Nazwa techniki cieni objętościowych wynika z faktu iż zasadniczą częścią algorytmu jest wyznaczenie brył obejmujących te fragmenty przestrzeni sceny, do których nie dochodzi światło tzn. brył, w których objętości panuje ciemność lub półcień (w przypadku gdy uwzględnia się wpływ światła rozproszonego). Utworzona w ten sposób dodatkowa geometria rysowana jest w buforze szablonowym w taki sposób aby dla każdego punktu obrazu otrzymać pewną liczbę na podstawie, której można będzie określić czy dany punkt jest oświetlony czy nie. Następnie informacja ta uwzględniana jest podczas wykonywania właściwego renderingu.

3.1. Algorytm

Rendering sceny z wykorzystaniem metody cieni objętościowych można wykonać na dwa sposoby. W pierwszym z nich obiekty znajdujące się w cieniu są rysowane z wykorzystaniem informacji o składowej oświetlenia otaczającego (ang. *ambient light*) co pociąga za sobą konieczność wykonania dwóch przebiegów renderingu całej sceny. Drugi sposób upraszcza całą procedurę gdyż scena renderowana jest tylko raz i dopiero po zakończeniu tego procesu odpowiednie obszary wynikowego obrazu zostają przyciemnione. Zyskuje się w ten sposób na szybkości działania jednak kosztem jakości zacienionych obszarów. Niemniej jednak w przypadkach gdy odpowiednio dobierze się stopień przyciemnienia różnica może być niezauważalna.

Schemat blokowy pierwszej z wyżej wymienionych wersji algorytmu przedstawia rysunek 7a. Rozpoczyna się on od wyznaczenia brył cieni. W tym celu wykorzystywane są informacje o geometrii sceny (współrzędne wierzchołków) oraz o położeniu źródła światła. Schemat wykonywanych w tym celu obliczeń zostanie szczegółowo przedstawiony w punkcie 3.2.



Rys. 7. Algorytm rysowania cieni objętościowych **a)** poprzez dwukrotne rysowanie sceny: w cieniu i w pełnym świetle lub **b)** poprzez przyciemnianie odpowiednich obszarów znajdujących się w cieniu.

Gdy zacienione fragmenty przestrzeni zostaną określone można przejść do właściwego renderingu. Pierwszy przebieg rysowania ma za zadanie wykonanie obrazu, na którym wszystkie obiekty znajdują się w cieniu. Oznacza to, że każdy obiekt jest oświetlony jedynie przez światło otaczające czyli takie, które nie pochodzi bezpośrednio ze swojego źródła lecz jest równomiernie rozproszone w całej przestrzeni. W przypadku stosowanego w OpenGL modelu Phonga ma ono stałą wartość dla całej sceny. W konsekwencji pierwszy przebieg renderingu wykonuje się wg. poniższego schematu.

1. Włączenie zapisu w buforze koloru, buforze głębokości oraz wyczyszczenie ich zawartości.
2. Włączenie oświetlenia składającego się jedynie ze światła otaczającego.
3. Narysowanie sceny.

Powyższe postępowanie można jeszcze uprościć poprzez całkowite wyłączenie oświetlenia i

w konsekwencji zaniechania zapisu do bufora kolorów (ponieważ i tak wszystkie rysowane obiekty byłyby czarne). W takim przypadku na końcowym obrazie cienie będą czarnymi obszarami. Można tak zrobić gdy ma się do czynienia z bardzo ciemnymi scenami gdzie wartość składowej światła rozproszonego jest równa lub bliska zeru. Tym co pozostaje w wyniku wykonania powyższych czynności i jest istotne dla kolejnego etapu renderingu jest zawartość bufora głębokości.

Zasadniczą częścią algorytmu jest określenie punktów obrazu, które są oświetlone. W tym celu dla każdego piksela można poprowadzić w głąb ekranu promień i zliczyć ilość jego przejść przez ściany brył cieni czyli ilość jego wejść i wyjść z cienia. Jeżeli liczby te są równe oznacza to, że dany punkt jest oświetlony (o ile obserwator nie znajduje się wewnątrz cienia). Dokonuje się tego poprzez odpowiednie rysowanie brył cieni w buforze szablonu. Szereg wykonywanych czynności w tym etapie renderingu przedstawiają poniższe punkty.

1. Wyłączenie zapisu do bufora kolorów oraz bufora głębi.
2. Ustawienie odpowiednich operacji na buforze szablonu, włączenie zapisu do niego oraz jego wyczyszczenie.
3. Narysowanie brył cieni.

Należy również wspomnieć, że test głębokości jest włączony i bazuje na zawartości z-bufora utworzonej podczas pierwszej fazy generowania obrazu. Dwa sposoby wykorzystania bufora szablonowego do wykonania powyższych czynności zostaną szczegółowo opisane w punkcie 3.3. Wyjściową informacją uzyskaną w wyniku powyższego postępowania jest pewna liczba określona dla każdego piksela determinująca fakt czy jest on w cieniu czy nie. Wartość 0 oznacza oświetlony punkt, a wartość różna od 0 zacieniony.

Ostatnim etapem tworzenia cieni jest oświetlenie odpowiednich fragmentów sceny. Dokonuje się tego poprzez ponowne narysowanie obiektów sceny, jednak tym razem w pełnym oświetleniu ograniczając jednocześnie rendering do tych punktów, dla których wartość w buforze szablonu jest równa 0. Szkic postępowania przedstawiają poniższe punkty.

1. Włączenie zapisu do bufora kolorów.
2. Wyłączenie zapisu do bufora szablonu oraz ustawienie testu szablonu aby dawał wynik pozytywny dla wartości równej 0.
3. Włączenie pełnego oświetlenia.
4. Narysowanie sceny.

Warto w tym miejscu również zaznaczyć, że ponowne rysowanie całej, w pełni oświetlonej

sceny jest mniej kosztowne niż w przypadku gdyby odbywało się samodzielnie. Dzieje się tak ze względu na fakt, że bufor głębokości jest już wypełniony odpowiednimi wartościami w rezultacie wykonania pierwszego przebiegu renderingu. Tak więc test głębokości przechodzą jedynie te fragmenty, które są widoczne na końcowym renderingu. Mówiąc inaczej, żaden z pikseli obrazu nie jest w tym momencie wielokrotnie nadpisywany.

Druga wersja algorytmu jest bardzo podobna, a zasadnicze różnice występują w etapach drugim i czwartym (rys. 7b). W etapie drugim wykonywany jest pełny rendering sceny uwzględniający wszystkie aspekty oświetlenia i materiałów. Oznacza to, że oprócz wpływu składowej otaczającej oświetlenia, znaczenie ma również działanie światła rozproszonego (ang. *diffuse*) oraz odbitego (ang. *specular*). Natomiast ostatnia faza tworzenia obrazu ogranicza się do narysowania półprzeźroczystego prostokąta zakrywającego cały ekran jednak zaktualizowane są tylko te piksele renderingu dla których wartość bufora szablonu wynosi 0. Podsumowując na etap czwarty składają się poniższe czynności.

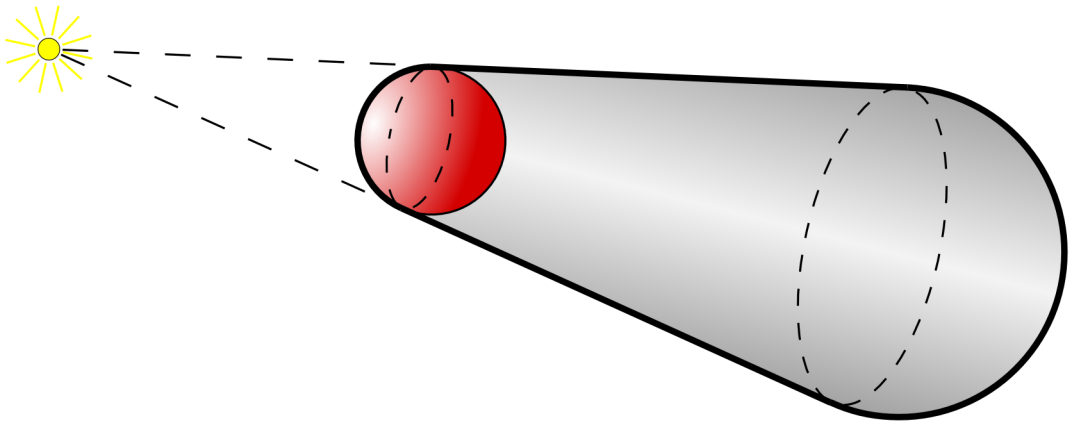
1. Włączenie zapisu do bufora kolorów.
2. Włączenie mieszania kolorów.
3. Wyłączenie zapisu do bufora szablonu oraz włączenie testu szablonu tak aby zwracał wynik pozytywny dla wartości 0.
4. Narysowanie półprzeźroczystego prostokąta, który przykryje cały ekran.

Rysowany prostokąt powinien pokrywać się z przednią płaszczyzną obcinania frusty (zob. dodatek C). Jeżeli jej położenie nie jest znane, można uzyskać identyczny efekt poprzez zastosowanie dwóch jednostkowych macierzy: projekcji oraz model-widok. Przy takiej konfiguracji górny-lewy róg prostokąta znajduje się w punkcie $(-1,1,0)$, a prawy-dolny w pozycji $(1,-1,0)$.

3.2. Wyznaczenie brył cieni

Istotnym elementem opisywanej w tym rozdziale metody jest wyznaczanie brył cieni. Jak zostało już wcześniej wspomniane każda taka bryła zawiera w swojej objętości pewną część przestrzeni sceny, do której nie dochodzą bezpośrednio promienie z danego źródła światła. Każda z brył konstruowana jest dla jednego obiektu oraz jednego światła, czyli niejako dla każdego rzucanego cienia osobno. Jej kształt wyznaczają promienie rzucane przez

światło przechodzące przez wierzchołki obrysu modelu widzianego z pozycji źródła światła rozpatrywanego obiektu. Przypomina on nieskończenie wysoki ostrosłup z obciętym wierzchołkiem. Powierzchnię tak wyznaczonej figury można podzielić na trzy fragmenty: ściany boczne, domknięcie przednie oraz tylne (rys. 8). Jej część boczną formuje się z czworoboków (lub trójkątów) stycznych do wspomnianych powyżej promieni, a jej przekrój poprzeczny ma kształt wspomnianego wcześniej obrysu obiektu widzianego z punktu źródła światła. Przekrój ten zwiększa swój rozmiar proporcjonalnie do odległości od światła. Domknięcie przednie (ang. *front cap* lub *light cap*) zamyka bryłę od strony światła i zazwyczaj tworzą je wielokąty tego fragmentu obiektu, których powierzchnia zwrócona jest w stronę źródła światła tzn. te, które są oświetlone. Domknięcie tylne (ang. *back cap* lub *dark cap*) pełni podobną rolę co powyższe jednak w przeciwieństwie do pozostałych jego dokładny kształt ma mniejsze znaczenie. Ważne jest jedynie to aby precyzyjnie łączyło się z częścią boczną bryły nie pozostawiając, dziur na jej powierzchni. Może ono powstać z odpowiednio przeskalowanej i przesuniętej w głąb sceny nie oświetlonej części obiektu lub w wyniku triangulacji wspomnianego wcześniej wielokąta tworzącego obrys bryły cienia. Domknięcie tylne może również w pewnym specyficznym przypadku, gdy padające na obiekt promienie są równoległe (tzn. punkt świetlny znajduje się w nieskończoności) przekształcić się w pojedynczy wierzchołek w nieskończoności (we współrzędnych jednorodnych). Warto w tym miejscu zaznaczyć, że bryła cienia nie zawsze musi być zamknięta tzn. może zostać pozbawiona jednego lub obu z wymienionych domknięć.



Rys. 8. Bryła cienia dla sfery. Linie przerywane zaznaczają kontury blokującego światło obiektu (z punktu widzenia światła). Pokazują również podział powierzchni bryły na trzy części (odpowiednio od lewej): domknięcie przednie, boki oraz domknięcie tylne.

Zanim przystąpi się do opisu właściwych obliczeń należałoby przybliżyć sposób w jaki, przedstawiona jest geometria sceny. Składają się na nią modele pojedynczych obiektów

utworzonych z powierzchni (ang. *faces*). W tym przypadku ich rolę pełnią pojedyncze trójkąty określone przez współrzędne swoich wierzchołków. Uściślając każdy obiekt jest siatką indeksowanych trójkątów tzn. opisany jest przez listę indeksów tworzących go trójkątów. W konsekwencji musi istnieć także lista trójkątów, do której odwołują się te indeksy. Każdy z trójkątów na wspomnianej liście zdefiniowany jest przez trzy indeksy wierzchołków. Ich kolejność jest zgodna z przyjętym kierunkiem nawijania, który standardowo w bibliotece OpenGL ma kierunek przeciwny do ruchu wskazówek zegara dla przednich stron trójkątów. Następną wykorzystywaną strukturą jest lista wierzchołków zdefiniowanych przez trzy (x , y , z) lub cztery współrzędne (x , y , z , w). Ponadto algorytm wymaga aby wszystkie obiekty były zamknięte tzn. żeby nie było na ich powierzchni dziur (ang. *cracks*). Oznacza to, że dany trójkąt przy każdej swojej krawędzi musi sąsiadować z innym trójkątem.

Każdy wierzchołek $\vec{V}_i$ opisany jest przez 3 lub 4 współrzędne. Wszystkie wierzchołki przechowywane są we wspólnym dla całego obiektu buforze w postaci tablicy, stąd do każdego z nich można odwołać się za pomocą odpowiedniego indeksu.

Struktura opisująca powierzchnię (trójkąt) jest nieco bardziej skomplikowana i zawiera następujące informacje:

- 3 indeksy (i_0 , i_1 , i_2) do tablicy wierzchołków zapisane zgodnie z kolejnością nawijania,
- czteroelementowy wektor normalny $\vec{P}$, którego składowe jednocześnie są współczynnikami równania tej płaszczyzny we współrzędnych jednorodnych.

Wektor normalny można obliczyć przy pomocy równania (30).

$$\vec{N}=(\vec{V}_{i_1}-\vec{V}_{i_0})\times(\vec{V}_{i_2}-\vec{V}_{i_0}) \tag{30}$$

Znając powyższe oraz jeden z punktów na tej płaszczyźnie czterowymiarowy wektor ją opisujący wyrażony jest następująco (31).

$$\vec{P}=(\vec{N}_x,\vec{N}_y,\vec{N}_z,-\vec{N}\cdot\vec{V}_{i_0}) \tag{31}$$

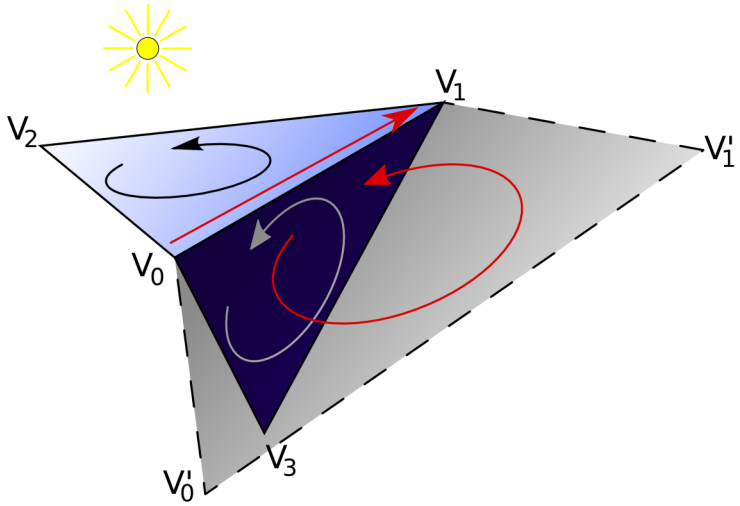
Zawarty w nim iloczyn skalarny jest odległością tej płaszczyzny od początku układu współrzędnych. Wszystkie powierzchnie podobnie jak wierzchołki przechowywane są w przeznaczonej dla nich liniowej tablicy.

Następną ważną strukturą jest krawędź zawierająca następujące pola:

- 2 indeksy do tablicy wierzchołków (i_0 , i_1),
- 2 indeksy do tablicy trójkątów (t_0 , t_1).

Istotne jest, aby kolejność punktów określających powyższy odcinek była zgodna z

kierunkiem nawijania pierwszego z trójkątów i jednocześnie przeciwna do kierunku nawijania drugiego z nich (por. rys. 9).



Rys. 9. Dwa sąsiadujące ze sobą trójkąty współdzielące ze sobą jedną ze swoich krawędzi. Prosta czerwona strzałka wskazuje jej kierunek. Owalne strzałki wskazują kierunek nawijania wierzchołków poszczególnych prymitywów. Półprzeźroczysty czworobok jest fragmentem części bocznej bryły cienia.

Analogicznie jak wspomniane wcześniej elementy opisu geometrii wszystkie krawędzie znajdują się we wspólnej tablicy jednak w przeciwieństwie do poprzednich nie jest ona wczytywana z pliku i należy ją samodzielnie utworzyć. W tym celu należy założyć, że dla każdej krawędzi pierwszy indeks wierzchołka i_0 będzie liczbą mniejszą od drugiego z nich i_1. Mając to na uwadze należy wykonać pętlę po całej tablicy trójkątów i dla każdego z nich dokonać porównania wartości kolejnych, bezpośrednio się poprzedzających indeksów wierzchołków (32).

$$\begin{aligned} i_0 &< i_1 \\ i_1 &< i_2 \\ i_2 &< i_0 \end{aligned} \tag{32}$$

Każda para przechodząca pozytywnie ten test tworzy nową krawędź. Fakt, że każda z nich sąsiaduje dokładnie z dwoma trójkątami o przeciwnych kierunkach gwarantuje, że krawędź zostanie tylko raz dodana do listy. W momencie dodawania odcinka do tablicy należy zapamiętać indeks pierwszej z przylegających do niego powierzchni. Aby uzyskać wymagany drugi wskaźnik należy wykonać powyższą pętlę ponownie. W tym przypadku warunek zostaje zmieniony i wyszukiwane zostają krawędzie o punktach w odwróconej kolejności. Całą sytuacje najlepiej zobrazuje poniższy pseudokod.

```
Dla każdego trójkąta T z listy powierzchni:
```

```
     Jeżeli dla wierzchołków spełnione jest i0 < i1 to:
          dodaj nową krawędź do lity krawędzi
          zapisz w krawędzi indeks pierwszego trójkąta T
     Jeżeli dla wierzchołków spełnione jest i1 < i2 to:
          dodaj nową krawędź do lity krawędzi
          zapisz w krawędzi indeks pierwszego trójkąta T
     Jeżeli dla wierzchołków spełnione jest i2 < i0 to:
          dodaj nową krawędź do lity krawędzi
          zapisz w krawędzi indeks pierwszego trójkąta T
Dla każdego trójkąta T z listy powierzchni:
     Jeżeli dla wierzchołków spełnione jest i0 > i1 to:
          Znajdź krawędź i0 -> i1
               zapisz w krawędzi indeks drugiego trójkąta T
     Jeżeli dla wierzchołków spełnione jest i1 > i2 to:
          Znajdź krawędź i1 -> i2
               zapisz w krawędzi indeks drugiego trójkąta T
     Jeżeli dla wierzchołków spełnione jest i2 > i0 to:
          Znajdź krawędź i2 -> i0
               zapisz w krawędzi indeks drugiego trójkąta T
```

Jak można zauważyć powyższy kod składa się z kilku pętli, które na szczęście są wykonywane tylko raz w czasie wczytywania siatki modelu z pliku. Podobnie jak wyliczona wcześniej normalna powierzchni informacje o krawędziach należą do statycznych danych obiektu i nie ulegają zmianom o ile obiekt nie zostanie poddany żadnym deformacjom.

Elementy sceny takie jak światło czy modele mogą zmieniać swoją pozycję oraz podlegać obrotom dlatego też następne obliczenia należy powtórzyć za każdym razem gdy powyższe przekształcenia mają miejsce. Do dalszych operacji na geometrii niezbędne będzie utworzenie tablicy zawierającej informacje o tym czy dana powierzchnia zwrócona jest w stronę światła. Dla każdego trójkąta przyporządkowuje ona wartość logiczną zapisaną pod odpowiednim indeksem zgodnym z jego pozycją w tablicy powierzchni. Do stwierdzenia tego faktu posłuży następująca nierówność:

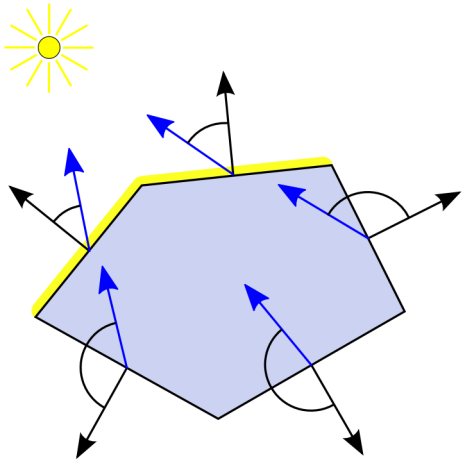
$$\vec{P} \cdot \vec{L}_o > 0 \,. \tag{33}$$

Jeżeli powyższy warunek jest prawdziwy, to trójkąt zwrócony jest w stronę światła. Przy

założeniu, że pierwsze trzy składowe równania płaszczyzny $\vec{P}$ tworzą jednostkowy wektor normalny, wynikiem powyższego iloczynu skalarnego we współrzędnych jednorodnych jest odległość punktu $\vec{L}_o$ od płaszczyzny. W niniejszym przypadku algorytm wymaga jedynie znajomości znaku otrzymanej wartości (czyli, wiedzy po której stronie płaszczyzny znajduje się źródło światła) dlatego wektor normalny niekoniecznie musi być jednostkowy. W trójwymiarowej przestrzeni euklidesowej nierówność (33) można zapisać inaczej:

$$\vec{N}\cdot(\vec{L}_o-\vec{V}_{i_0})>0\,, \tag{34}$$

gdzie $\vec{N}$ jest wektorem normalnym płaszczyzny, a różnica $\vec{L}_o-\vec{V}_{i_0}$ jest kierunkiem promienia światła przechodzącego przez wierzchołek $\vec{V}_{i_0}$. Ponieważ wszystkie wierzchołki $(\vec{V}_{i_0},\vec{V}_{i_1},\vec{V}_{i_2})$ leżą w tej samej płaszczyźnie oraz istotny jest tylko znak powyższego iloczynu to do wyznaczenia promienia można wykorzystać dowolny z nich. Interpretację iloczynu skalarnego (33) przedstawia rysunek 10.



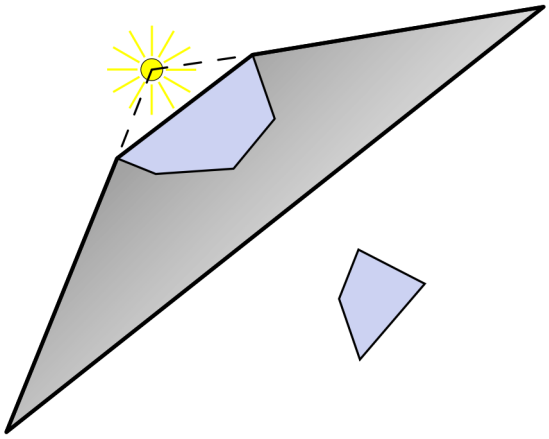
Rys. 10. Test oświetlenia powierzchni. Czarne strzałki są wektorami normalnymi powierzchni zaś niebieskie to lokalne dla danej płaszczyzny wektory kierunku światła (równoległe do promieni).

Posługując się postacią (34) nierówności należy uwzględnić dodatkowo przypadek gdy światło znajduje się w nieskończoności ($\vec{L}_{o_w}=0$):

$$\vec{N}\cdot\vec{L}_o>0\,. \tag{35}$$

Pominięcie w wyrażeniu (35) położenia wierzchołka $\vec{V}_{i_0}$ wynika bezpośrednio z faktu, że w przypadku różnicy wektorów (por. wzór (64)) w przestrzeni jednorodnej wektor w nieskończoności bezpośrednio determinuje wynik tego działania. Reasumując można stwierdzić, że podejście (33) jest bardziej elastyczne ponieważ uwzględnia oba warianty.

Podczas wykonywania obliczeń należy mieć na uwadze również fakt, że wektor położenia źródła światła $\vec{L}_o$ musi być wyrażony we współrzędnych lokalnych rozpatrywanego obiektu (współrzędnych obiektu – zob. dodatek C). W przypadku gdy wierzchołki modelu poddawane były transformacjom macierzowym, które przeważnie wykonywane są sprzętowo przez OpenGL szybciej jest przetransformować pojedynczy punkt źródła światła $\vec{L}$ do przestrzeni obiektu niż wszystkie wierzchołki danego modelu (powielając tym samym obliczenia OpenGL) do współrzędnych świata. Powyższe przekształcania można wykonać poprzez zastosowanie macierzy odwrotnej do macierzy transformacji modelu na wektorze położenia źródła światła. Podsumowując opisywany etap algorytmu aby utworzyć opisywaną tablicę należy przejść po całej liście powierzchni wykonując jednocześnie dla każdej z nich iloczyn skalarny jej wektora normalnego $\vec{P}$ z wektorem położenia światła $\vec{L}_o$ po to aby ostatecznie zapamiętać wartość logiczną będącą wynikiem nierówności (33), która określa czy dana płaszczyzna jest oświetlona czy zacieniona.



Rys. 11. Skończona bryła cienia powstała poprzez wytłoczenie obrysu obiektu na ustaloną z góry odległość. Na ilustracji widać, że nie pokrywa ona jednego z obiektów sceny.

W momencie gdy znane są powyższe wartości można przystąpić do wyznaczenia bryły cienia. Wyróżnia się dwa rodzaje brył cieni w zależności od ich długości. Może ona być skończona czyli jej boki posiadają ustaloną stałą długość lub nieskończona co oznacza, że jej domknięcie tylne znajduje się w nieskończoności. Pierwsza z powyższych wariancji ma niestety poważna wadę, która ujawnia się zazwyczaj gdy światło jest bardzo blisko obiektu. W takim przypadku utworzona bryła może nie zawierać w swojej objętości wszystkich zacienionych obiektów (rys. 11). Drugi rodzaj ma również pewną przypadłość jaką jest przycinanie bryły przez frustę (zob. dodatek C), a konkretnie przez tylną jej ścianę (ang. *far*

plane clip). Może to powodować błędy na ostatecznym renderingu. Rozwiązaniem okazuje się użycie odpowiedniej macierzy projekcji, w której tylna płaszczyzna frusty umieszczona jest w nieskończonej odległości od obserwatora. Szczegółowy opis wyprowadzenia wspomnianej macierzy znajduje się w punkcie 3.4. Bez względu na fakt, który rodzaj bryły się wybierze dalsze postępowanie dla obu przypadków jest podobne.

Jak zostało już wcześniej wspomniane, najważniejszą częścią powierzchni bryły cieni są jej boki. Pierwszym krokiem do wyznaczenia ich kształtu jest określenie sylwetki obiektu widzianego z pozycji źródła światła (linia przerywana na czerwonym obiekcie z rys. 8). Obrys ten składa się z pojedynczych odcinków wybranych z utworzonej wcześniej pełnej listy krawędzi. Selekcji dokonuje się poprzez przeglądniecie wspomnianej tablicy w poszukiwaniu odcinków współdzielonych przez trójkąty, z których jeden jest oświetlony, a drugi nie (rys. 9). Jeżeli taka krawędź zostanie znaleziona należy utworzyć wzdłuż niej czworobok określony przez należące do niej dwa wierzchołki oraz kolejne dwa powstałe z ich przesunięcia zgodnie z kierunkiem padania promieni świetlnych (punkty $\vec{V}'_0$ i $\vec{V}'_1$ z rys. 9). Dla nieskończonej bryły cienia punkty $\vec{V}'_i$ oblicza się wg wzoru (36).

$$\vec{V}'_i = \vec{V}_i - \vec{L} \tag{36}$$

Po wykonaniu powyższej różnicy należy zastąpić współrzędną V'_{iw} wartością 0 (po uprzednim przekształceniu tego wektora do współrzędnych euklidesowych gdzie $V'_{iw}=1$). Przy założeniu, że $L_w=1$ i $V_{iw}=1$ obie operacje można uprościć do postaci z równania (37).

$$\vec{V}'_i = (V_{ix}-L_x, V_{iy}-L_y, V_{iz}-L_z, 0) \tag{37}$$

Warto zauważyć, że powyższe obliczenia mają zastosowanie jedynie dla przypadku punktowego źródła światła, którego współrzędna $L_w \neq 0$. W przeciwnym razie, tzn. gdy znajduje się ono w nieskończoności $L_w=0$, odcinki (a ściślej półproste) $V_0V'_0$ i $V_1V'_1$ są równoległe. Oznacza to, że punkty V'_0 i V'_1 zajmują to samo położenie w nieskończoności. Wówczas czworoboki części bocznej bryły cienia można zastąpić trójkątami o wspólnym wierzchołku V' (38).

$$\vec{V}' = -\vec{L} \tag{38}$$

W przypadku skończonej bryły cienia, której boki mają stałą długość (rys. 11) postępuje się podobnie, ale w tym przypadku należy wykonać dodatkowe operacje. Po pierwsze konieczne jest znormalizowanie wektora $\vec{V}'_i$ z równania (36). Następnie należy go przeskalować mnożąc wszystkie jego składowe przez założoną z góry liczbę określającą długość boku

bryły l:

$$\vec{V}'_i = \frac{\vec{V}_i - \vec{L}}{\|\vec{V}_i - \vec{L}\|} l \tag{39}$$

Tworzenie domknięcia przedniego (ang. *front cap* lub *light cap*) należy do najprostszej części formowania objętości cienia. Tworzą je trójkąty, które w liście oświetlonych powierzchni są zaznaczone jako te, które są skierowane w stronę źródła światła. Ich wierzchołki nie podlegają żadnym dodatkowym transformacjom dlatego też wystarczy je po prostu narysować.

Domknięcie tylne (ang. *back cap* lub *dark cap*) analogicznie jak przednio tworzy ta część geometrii obiektu, która w tablicy jest zaznaczona jako nieoświetlona. Jednakże wierzchołki zakwalifikowanych do niej trójkątów należy poddać przekształceniom (36) lub (37), takim samym jak te transformujące punkty V_i w V'_i dla części bocznej bryły. Warto tutaj również przypomnieć, że w przypadku źródła światła umieszczonego w nieskończonej odległości od obiektu rysowanie domknięcia tylnego nie jest konieczne ponieważ wierzchołki V'_i reprezentują jedną i tę samą pozycję w przestrzeni V'.

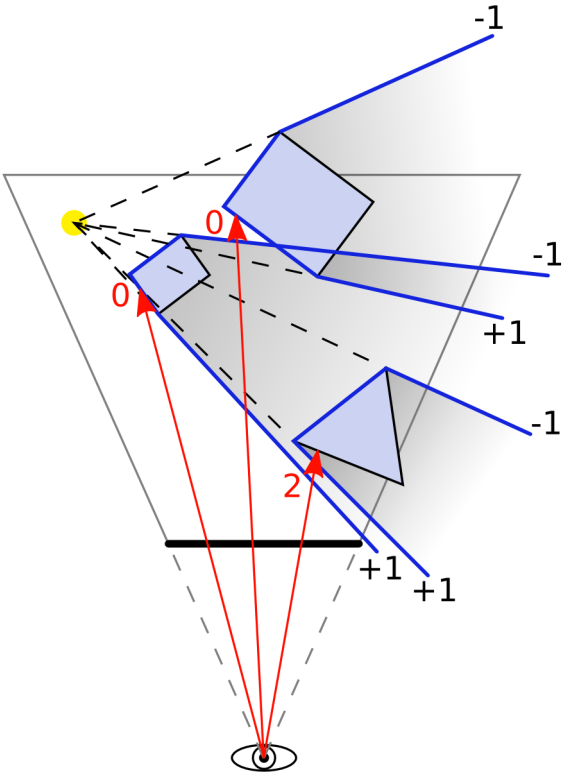
3.3. Odmiany operacji na buforze szablonu

Jeżeli znana jest już geometria brył cieni oraz zawartość bufora głębokości można przystąpić do zaznaczenia tych obszarów obrazu, które znajdują się w cieniu. W tym celu wykorzystywany jest bufor szablonowy. Powszechnie stosowane są dwa sposoby jego zastosowania *depth pass* (*z-pass*) oraz *depth fail* (*z-fail*). Oba mają swoje zalety i wady i trudno jednoznacznie stwierdzić, który z nich jest lepszy.

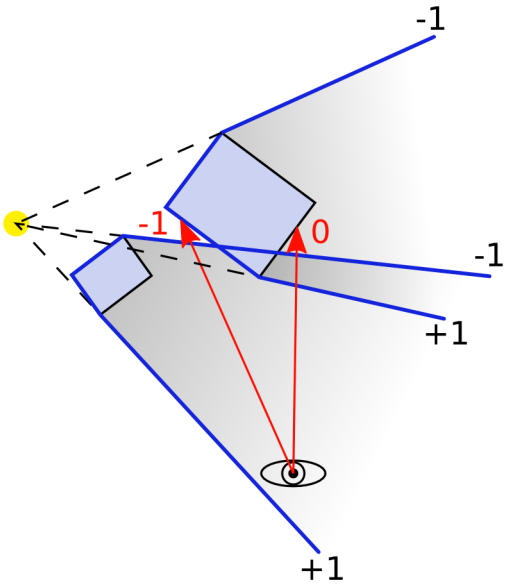
Algorytm wykorzystuje fakt, że każdy z rysowanych trójkątów posiada swoją przednią i tylną stronę określoną przez kierunek nawijania (rys. 9). Dla domyślnych ustawień OpenGL oznacza to, że jeżeli wierzchołki danego trójkąta po rzutowaniu na ekran występują w kolejności przeciwnej do ruchu wskazówek zegara widoczna jest przednia część trójkąta. Fakt ten wykorzystywany jest przez bibliotekę m.in. do ograniczenia ilości wypełnianych pikseli (ang. *pixel fillrate*) podczas rasteryzacji obrazu. Oczywiste jest, że w przypadku zamkniętego obiektu jego wewnętrzna strona nie jest widoczna dlatego jej rendering byłby nadmiarowy. Bez takiej funkcjonalności narysowana wewnętrzna część bryły zostałaby i tak przysłonięta

przez kolejny renderowany wielokąt. OpenGL oferuje możliwość wyłączenia widoczności wybranej ze stron, którą domyślnie jest strona tylna (stąd ang. *back face culling*). Bryły cieni zostały wygenerowane w taki sposób, aby przednia strona ich powierzchni skierowana była na zewnątrz.

Wersja algorytmu *depth pass* (*z-pass*) zawdzięcza swoją nazwę temu, że przy zliczaniu przejść promienia przez bryły cieni brane są pod uwagę tylko te z nich, które znajdują się przed widocznymi obiektami sceny (czyli same byłyby widoczne gdyby zostały narysowane w buforze kolorów). Wartości w szablonie zmieniane są tylko dla tych fragmentów, dla których test głębokości zwrócił rezultat pozytywny, czyli gdy ich odległość od ekranu jest mniejsza od wartości zapisanej w z-buforze. Oznacza to, że w tym przypadku nie jest wymagane domknięcie przednie ponieważ pokrywa się ono z oświetloną częścią modelu tzn. jest niewidoczne. Działanie algorytmu ilustruje rysunek 12.



Rys. 12. Działanie algorytmu *stencil pass* (*z-pass*). Czerwone strzałki oznaczają promienie wyprowadzone z danych pikseli obrazu wgłąb ekranu (gruba czarna kreska). Liczby na ich końcach przedstawiają wartość zapisaną w buforze szablonu. Grube niebieskie kontury są przekrojami przez bryły cieni.



Rys. 13. Błędne działanie algorytmu *stencil pass* (*z-pass*) w przypadku gdy obserwator znajduje się wewnątrz cienia.

Wejście promienia w cień oznacza dodanie 1 do wartości w szablonu, a jego opuszczenie

odejmuje 1. Jeżeli otrzymana w wyniku tych przejść liczba jest równa 0 dany piksel jest oświetlony w przeciwnym wypadku jest zacieniony. W praktycznej implementacji zliczanie realizuje się m.in. poprzez wykorzystanie opisywanej wcześniej możliwości rozróżnienia tylnych oraz przednich stron trójkątów i co za tym następuje ukrywania odpowiedniej z nich. Narysowanie przedniej strony każdego z trójkątów (widok fragmentu zewnętrznej powierzchni bryły cienia) odpowiada wejściu w cień, natomiast narysowanie tylnej części każdego z trójkątów (widok fragmentu wewnętrznej powierzchni bryły) odpowiada wyjściu z cienia. Dzięki temu można określić czy promień wchodzi czy wychodzi z cienia. Algorytm można zapisać w następujących punktach:

1. Wyłączenie zapisu do bufora kolorów oraz do z-bufora.
2. Ustawienie testu głębokości tak aby dawał rezultat pozytywny dla wartości mniejszych niż te zapisane w buforze.
3. Ustawienie bufora szablonu na inkrementację wartości dla fragmentów, które przeszły pozytywnie powyższy test.
4. Włączenie ukrywania tylnych stron trójkątów.
5. Narysowanie brył cieni.
6. Zmiana sposobu wypełniania bufora szablonu na dekrementację.
7. Zmiana ukrywania stron trójkątów na stronę przednią.
8. Narysowanie brył cieni.

Ponieważ bryły cieni nie mają być widoczne na ostatecznym renderingu celem pierwszego punktu jest zapewnienie takiego właśnie zachowania biblioteki. Krok 2 gwarantuje, że będą rozpatrywane tylko widoczne fragmenty brył cieni, czyli te nie zasłonięte przez żadne obiekty sceny. Następne działania posłużą właściwym operacjom na buforze szablonu. Punkt 4 sprawia, że wnętrze brył cieni nie będzie widoczne. W powiązaniu z poprzedzającym go ustawieniem inkrementacji bufora szablonu pozwala to na zliczenie wejść promieni w bryły cieni. W kolejnych punktach sytuacja zostaje odwrócona. Krok 7 zmienia sposób rysowania tak, że widoczne są wewnętrzne ściany brył. Każdy widoczny fragment takiej powierzchni zgodnie z ustawieniami z punktu 6 powoduje dekrementacje wartości w buforze. W ten sposób uwzględniona zostaje liczba wyjść promieni z obszarów cieni.

Przedstawiony powyżej algorytm ma niestety pewną poważną wadę. O ile w przypadku przedstawionym na rysunku 12 daje poprawne rezultaty całość postępowania załamuje się w momencie gdy obserwator (ekran) znajduje się wewnątrz cienia (rys. 13). Jednym z

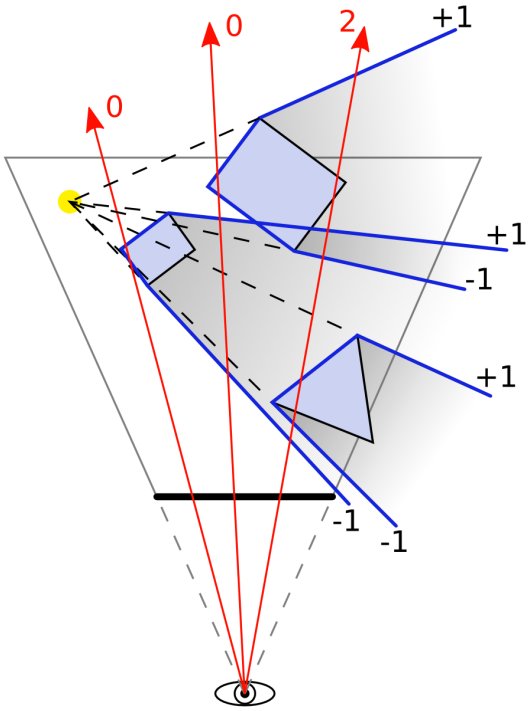
pierwszych proponowanych sposobów zapobiegania temu niekorzystnemu zjawisku było domknięcie bryły w miejscu przecięcia się jej z przednią płaszczyzną przycinania (ekranem). Rozwiązanie takie zostało zaprezentowane jest m.in. w publikacji [5] i polega na rzutowaniu domknięcia tylnego bryły na powierzchnie ekranu. Jednak błędy numeryczne mogą powodować wystąpienie artefaktów na wynikowym obrazie (np. szczelin pomiędzy trójkątami).

W celu rozwiązania problemu obserwatora znajdującego się w cieniu powstała druga wersja algorytmu nazywana *depth fail* (*z-fail*), której działanie zostało przedstawione na rysunku 14. W tym przypadku postępowanie jest podobne jak w metodzie *depth pass*, jednak pewne czynności zostały odwrócone. Tym razem brane pod uwagę są te fragmenty brył cieni, które gdyby spróbować je narysować nie byłyby widoczne na końcowym renderingu ponieważ znajdowałyby się za lub pokrywałyby się z widocznymi obiektami sceny. Jak można się domyślić implikuje to konieczność zastosowania domknięć przednich jak i tylnych brył utworzonych z oświetlonych części powierzchni obiektów rzucających cień. Zmianie ulegają również operacje na szablonie. Tym razem wejście w cień zmniejsza wartość w buforze natomiast wyjście – zwiększa ją. Uwzględnienie powyższych modyfikacji prowadzi do wykonania wymienionych poniżej czynności.

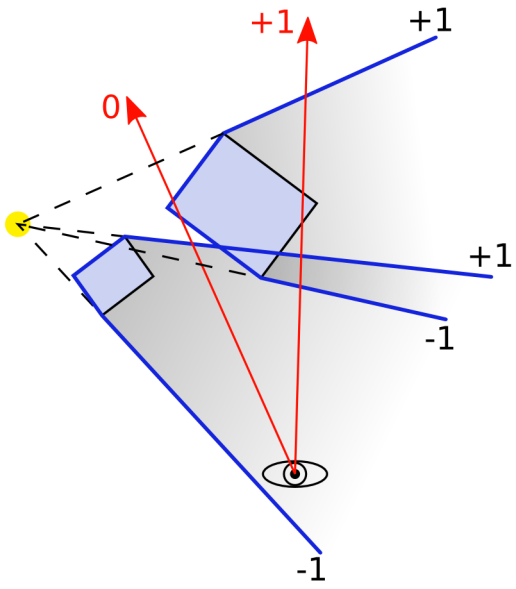
1. Wyłączenie zapisu do bufora kolorów oraz do z-bufora.
2. Ustawienie testu głębokości tak aby dawał rezultat pozytywny dla wartości mniejszych od tych zapisanych w buforze.
3. Ustawienie bufora szablonu na inkrementację wartości dla fragmentów, które nie przeszły pozytywnie powyższego testu.
4. Włączenie ukrywania przednich stron trójkątów.
5. Narysowanie brył cieni.
6. Zmiana sposobu wypełniania bufora szablonu na dekrementację.
7. Zmiana ukrywania stron trójkątów na stronę tylną.
8. Narysowanie brył cieni.

W stosunku do poprzedniego przepisu dla wersji *depth pass* przebudowie uległy punkty 3, 4, 6 oraz 7. Pierwszy zmodyfikowany krok zmienia sposób dokonywania operacji na wartościach szablonu tak aby zmianie ulegały wartości odpowiadające tym fragmentom, które nie przeszły testu głębokości. Analogicznej modyfikacji została poddana czynność z pozycji 6. Dodatkowo w pierwszej kolejności rysowane są wnętrza brył cieni zwiększając tym samym

wartość szablonu, a dopiero po nich zewnętrzne ściany dekrementujące tę liczbę.



Rys. 14. Prezentacja działania algortymu *stencil fail* (*z-fail*). Niebieskie kontury odpowiadają przekrojom przez bryły cieni. Czerwonymi strzałkami zostały zaznaczone promienie prostopadłe do ekranu odpowiadające wybranym pikselom obrazu, a liczby na ich końcach oznaczają wartość zapisaną w buforze szablonu.



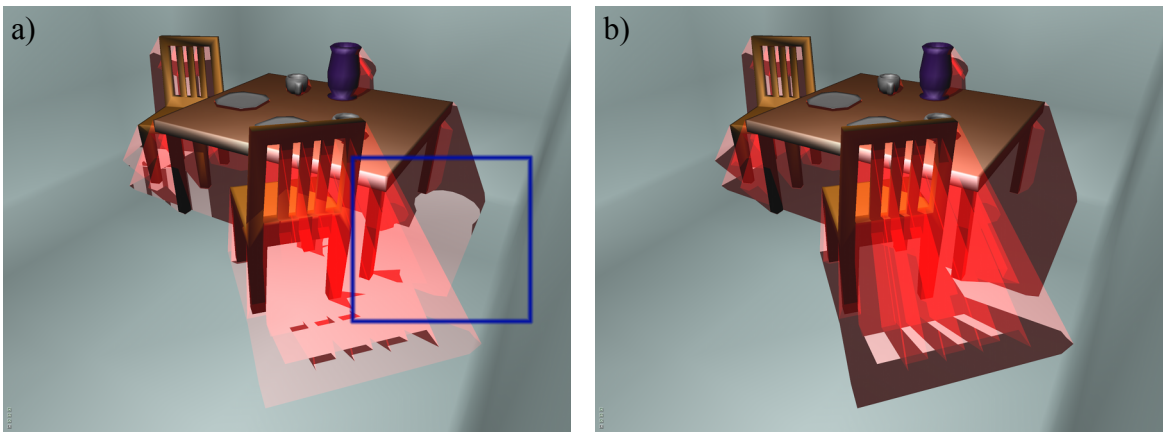
Rys. 15. Ilustracja poprawności działania algorytmu *stencil fail* (*z-fail*) dla przypadku, w którym obserwator znajduje się wewnątrz cienia.

Jak zostało wcześniej wspomniane zaletą powyższego rozwiązania jest to, że radzi sobie ono z przypadkiem gdy obserwator znajduje się w cieniu (rys. 15). Kosztem jaki należy za to ponieść może być zwiększona ilość rysowanych pikseli. W poprzedniej wersji algorytmu rozpatrywane były tylko widoczne bryły cieni, zaś w tym przypadku uwzględniane muszą być tylko te ukryte, których zazwyczaj jest dużo więcej. Dochodzi do tego konieczność rysowania domknięcia przedniego. W zależności od złożoności obiektu może mieć ono postać dość skomplikowanej siatki w przeciwieństwie do części bocznej składającej się wyłącznie z prostokątów w liczbie równej ilości odcinków krawędzi obrysu obiektu. Cena ta może jednak być do zaakceptowania w zamian za pewność otrzymania prawidłowych rezultatów. Drugim mankamentem metody *depth fail* w przypadku zastosowania nieskończonych brył cieni jest przycinanie ich przez frustę. W konsekwencji powstaje niedomknięta bryła generująca błędne wartości w buforze szablonowym (rys. 16). Dla wersji *depth pass* nie ma to znaczenia ponieważ w tym przypadku brane pod uwagę są tylko widoczne bryły. Oznacza to, że w części obrazu gdzie to ma miejsce obiekty znajdują się za przyciętymi bryłami, więc same

również są wykluczone z renderingu. W konsekwencji nie zostaną narysowane powierzchnie, na których owy błędny cień miałby się pojawić. Rozwiązaniem powyższego problemu dla *depth fail* okazuje się rezygnacja przycinania obiektów przez tylną płaszczyznę obcinania. Dokonuje się tego przez zastosowanie odpowiedniej macierzy projekcji opisanej w następnym punkcie tej pracy (zob. [10]).

3.4. Nieskończenie odległa tylna płaszczyzna obcinania

Poważnym problemem jaki napotyka się przy implementacji techniki cieni objętościowych jest obcinanie przez frustę brył zawierających nieoświetlone obszary. W rezultacie ich powierzchnia nie spełnia jednego z założeń algorytmu mówiącego, że musi ona być zamknięta czyli nie może posiadać dziur. Dla wersji *depth pass* algorytmu źródłem problemu jest płaszczyzna przednia frusty, dla *depth fail* – płaszczyzna tylna. O ile dla pierwszego przypadku nie ma dobrej recepty, jednocześnie wydajnej oraz dającej zadowalające rezultaty (zob. punkt 3.3), to dla drugiej istnieje pewne dobre rozwiązanie. Polega ono na umieszczeniu tylnej płaszczyzny obcinającej frusty w nieskończonej odległości od obserwatora (rys. 16).



Rys. 16. Błędy wynikające z obcinania brył cienia przez frustę. Na obrazku **a)** został zaznaczony fragment bryły cienia wazonu, która obcięta powoduje błędy przy obliczaniu wartości w szablonie. Zastosowanie nieskończenie odległej frusty lub skończonych brył **b)** rozwiązuje problem.

Wiąże się to z koniecznością zmodyfikowania standardowej macierzy projekcji omówionej szczegółowo w dodatku C. Parametr f macierzy (84) określa odległość tylnej płaszczyzny obcinającej. Aby umieścić ją w nieskończoności należy przejść do granicy tej macierzy, w której f dąży do nieskończoności (40).

$$M_{\infty}(l,r,b,t,n)=\lim_{f\to\infty} M_{proj}(l,r,b,t,n,f)=\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (40)$$

Dokonując analogicznego mnożenia, jakie ma miejsce w równaniu (85) iloczyn wektora położenia wierzchołka $\vec{V}_e$ wyrażonego w układzie współrzędnych oka (ang. *eye coordintes*) i powyższej macierzy projekcji daje w rezultacie punkt we współrzędnych przycięcia (ang. *clip coordinates*) $\vec{V}_c$.

$$\vec{V}_c=M_{\infty}\cdot\vec{V}_e=\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix}\cdot\begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix}=\begin{bmatrix} \frac{2nx_e+(r+l)y_e}{r-l} \\ \frac{2ny+(t+b)z}{t-b} \\ -z_e-2nw_e \\ -z_e \end{bmatrix}=\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} \quad (41)$$

Zakładając $w_e>0$ ($w_e<0$ powodowałoby odwrócenie kierunku wektora) można zauważyć, że wartość otrzymanej współrzędnej z_c będzie zawsze mniejsza od wartości w_c. Z tego właśnie powodu trzecia z nierówności (86) dla punktów znajdujących się przed obserwatorem będzie zawsze prawdziwa. Tym samym wierzchołki w nieskończoności o $w_e=0$ również nigdy nie zostaną odrzucone (wtedy $z_d=w_d$), a mający następnie miejsce podział perspektywiczny (zob. równanie (87)) zwróci wartość $z_d=1$.

Na tę chwilę może się wydawać, że założony cel został osiągnięty. Niestety sprzętowe błędy wynikające z niedostatecznej precyzji liczb mogą doprowadzić do tego iż wartość z_d będzie nieznacznie większa od jedności. Skutkuje to odrzuceniem danego wierzchołka podczas operacji obcinania czego uniknięcie było przecież celem przy powyższej modyfikacji macierzy projekcji. Pomysłem rozwiązującym ów problem jest zmiana przedziału w jakim ma się docelowo znaleźć współrzędna z_d wierzchołka w znormalizowanych współrzędnych urządzenia. Jak zostało wspomniane w dodatku C standardowo zakres ten zawiera się pomiędzy -1 i 1. Należy więc zapewnić pewien bezpieczny margines błędu ϵ uzyskując tym samym przedział $\langle -1, 1-\epsilon \rangle$. Tym samym szerokość tego zakresu zmalała z wartości 2 do $2-\epsilon$. Mając to na uwadze przekształcenie współrzędnej z_d tak, aby znalazła się w nowym przedziale dokonuje się przy pomocy działania (42).

$$z'_d=(z_d+1)\frac{2-\epsilon}{2}-1 \tag{42}$$

Powyższa transformacja wyrażona jest poprzez znormalizowane współrzędne urządzenia. Celem jednak jest uzyskanie nowej macierzy projekcji, której zastosowanie przekształca wektor ze współrzędnych oka do współrzędnych przycięcia. Należy więc podstawić do równania (42) ostatni wiersz macierzy (87) (czyli przepisu na obliczenie z_d):

$$\frac{z'_c}{w_c}=\left(\frac{z_c}{w_c}+1\right)\frac{2-\epsilon}{2}-1\ . \tag{43}$$

Po podstawieniu wartości współrzędnych z_c oraz z_w otrzymanych w wyniku operacji (41) powyższa zależność przyjmuje postać (44).

$$\frac{z'_c}{-z_e}=\left(\frac{-z_e-2n\,w_e}{-z_e}+1\right)\frac{2-\epsilon}{2}-1 \tag{44}$$

Stąd ostatecznie można wyznaczyć z'_c:

$$z'_c=-(\epsilon-1)z_e+n(\epsilon-2)w_e\ . \tag{45}$$

Wiedząc, że wartość składowej z_c wektora $\vec{V}_c$ powstaje z mnożenia 3 kolumny macierzy (40) i wektora $\vec{V}_e$ można tak zmodyfikować tę macierz aby wynikowy wektor miał 3 składową w postaci (45). Zmiana ta przedstawiona jest poniżej:

$$M_\infty(l,r,b,t,n,\epsilon)=\begin{bmatrix}\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0\\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0\\ 0 & 0 & \epsilon-1 & n(\epsilon-2)\\ 0 & 0 & -1 & 0\end{bmatrix}. \tag{46}$$

Opisane rozwiązanie pozwala osiągnąć zamierzony cel jednak warto w tym miejscu zadać pytanie o to, jak przesunięcie tylnej płaszczyzny obcinania frusty wpływa na dokładność bufora głębokości. W tym celu można określić funkcję, $d_{proj}(P)$ (47) wykorzystującą macierz M_{proj} (84) do mapowania współrzędnej z_e punktu $P_e=(x_e,y_e,z_e,1)$ wyrażonej we współrzędnych oka jako znormalizowanej współrzędnej urządzenia:

$$d_{proj}(P_e)=\frac{z_c}{w_c}=\frac{-(f+n)z_e+2f\,n\,w_e}{-z_e(f-n)}=\frac{f+n}{f-n}+\left(\frac{2f\,n}{f-n}\right)\frac{1}{z_e} \tag{47}$$

Analogiczna funkcję można wyznaczyć dla nieskończonej macierzy M_∞ (40):

$$d_\infty(P_e)=\frac{z_{e\infty}}{w_{e\infty}}=\frac{-z_e-2nw_e}{-z_e}=1+2n\frac{1}{z_e}\,. \tag{48}$$

Odległość dwóch punktów P_1 i P_2 dla macierzy M_{proj} wynosi więc:

$$d_{proj}(P_2)-d_{proj}(P_1)=\frac{2nf}{f-n}\left(\frac{1}{z_2}-\frac{1}{z_1}\right), \tag{49}$$

a dla macierzy M_∞:

$$d_\infty(P_2)-d_\infty(P_1)=2n\left(\frac{1}{z_2}-\frac{1}{z_1}\right). \tag{50}$$

Równania (49) i (50) pokazują, że w przypadku zastosowania nieskończenie odległej tylnej płaszczyzny obcinającej przedział pomiędzy wierzchołkami jest mniejszy o mnożnik $\frac{f}{f-n}$. W praktycznych zastosowaniach liczba n jest o wiele mniejsza od 1, a liczba f dużo większa od 1. Oznacza to, że wartość powyższego współczynnika jest bliska 1. Tym samym utrata dokładności z-bufora jest niewielka.

Metoda cieni objętościowych zyskała sporą popularność co bezpośrednio przełożyło się na wsparcie oferowane dla niej przez producentów sprzętu. Dla przykładu firma NVidia zaproponowała rozszerzenie OpenGL o nazwie `GL_NV_depth_clamp` pozwalające wyeliminować odrzucanie prymitywów znajdujących się przed przednią lub za tylną płaszczyzną obcinania (zob. [6]). Tym samym zostają one „przytwierdzone” (ang. *clamp*) do wspomnianych powierzchni. Realizowane jest to poprzez ustawienie współrzędnej z_e wyrażonej w układzie współrzędnych znormalizowanych urządzenia na -1 (dla prymitywów przed przednią płaszczyzną obcinania) lub 1 (dla wierzchołków za frustą). Dzięki temu powyższe rozszerzenie pozwala rozwiązać opisywany problem w bardzo wygodny sposób. Po włączeniu dodanej przez nie zmiennej stanu OpenGL funkcją `glEnable(GL_DEPTH_CLAMP_NV)` obiekty rysowane będą zgodnie z opisywanymi tutaj zasadami. W takim razie użycie macierzy M_∞ nie jest już wymagane.

3.5. Udoskonalenia i optymalizacje

Praktyczne implementacje algorytmu tworzenia cieni objętościowych w komercyjnych aplikacjach oraz w grach są przeważnie bardziej skomplikowane. Zwłaszcza w drugim

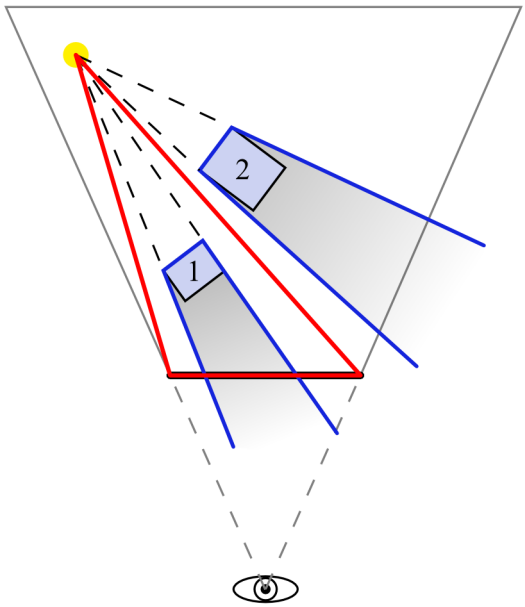
wymienionym przypadku szybkość rysowania cieni jest bardzo istotna. Uzyskanie szybkości animacji rzędu 30 – 60 klatek na sekundę uznaje się za konieczne co oznacza, że aby spełnić powyższe założenie jedna ramka obrazu musi zostać wygenerowana w czasie od ok. 16 do 33ms. Należy przy tym zaznaczyć, że tworzenie cieni nie jest jedynym zadaniem obliczeniowym w tego typu aplikacjach.

Najprostszym sposobem przyspieszenia algorytmu jest ponowne sięgnięcie po rozwiązania sprzętowe oferowane przez karty graficzne. Jednym z nich jest wykorzystanie oddzielnych funkcji szablonu dla przednich i tylnych stron trójkątów. Dzięki temu bryły cieni mogą być rysowane tylko raz. W takim przypadku modyfikacja algorytmu *depth pass* polega na wyłączeniu ukrywania niewidocznych stron trójkątów (czyli ich jednoczesnym rysowaniu) oraz ustawieniu operacji na szablonie w taki sposób aby przednia strona powierzchni bryły inkrementowała wartość w szablonie natomiast tylna, wewnętrzna dekrementowała. W wersji *depth-fail* ma miejsce odwrotna sytuacja tzn. przednie strony trójkątów zmniejszają, a tylne zwiększają liczbę zapisaną w buforze szablonu. Możliwość ustawienia odmiennych funkcji dla operacji na szablonie dostępna jest gdy sterownik OpenGL obsługuje rozszerzenie `GL_EXT_stencil_two_side` (zob. [7]) lub jest zgodny z wersją 2.0 biblioteki (zob. [4]).

Powyższe rozwiązanie nie gwarantuje niestety, że wynik będzie prawidłowy. Może się tak zdarzyć, że gdy wartość w buforze wynosi 0, będzie ona najpierw dekrementowana. W takim przypadku pozostanie ona nadal równa 0, a po następującej po niej inkrementacji będzie równa 1 co jest wartością nieprawidłową. OpenGL od wersji 1.4 oraz te implementacje, które obsługują rozszerzenie `GL_EXT_stencil_wrap` (zob. [8]) pozwala na „zawijanie” wartości, gdy przekracza ona swój dozwolony zakres (nadaje się jej minimalną dozwoloną wartość, gdy przekracza maksymalną dozwoloną i odwrotnie). Przy wykorzystaniu bufora szablonu o głębokości 8 bitów dostępne wartości mieszczą się w zakresie od 0 do 255, co w omawianym powyżej przypadku, jeżeli zastosuje się „zawijanie” wartości, pozwoli zamiast nieprawidłowego 0 uzyskać liczbę 255. Następująca po niej inkrementacja zwróci oczekiwany wynik równy liczbie 0. Ponadto warto również rozpatrzyć inną ewentualność mogącą wystąpić również bez zastosowania oddzielnych funkcji szablonu. Istnieje szansa, że ilość tych samych stron trójkątów przekroczy 255. W tym przypadku zastosowanie wspomnianego rozszerzenia „zawijającego” wartości bufora szablonu umożliwia uniknięcie występujących w takim przypadku błędów.

Jedną z kolejnych optymalizacji zaproponowanych m.in. w pozycji [9] jest stosowanie

algorytmów *depth pass* oraz *depth fail* równocześnie i wybieranie pomiędzy nimi w zależności od wzajemnej pozycji obserwatora, źródła światła oraz obiektu rzucającego cień. Celem jest wykorzystywanie wersji *depth pass* w każdym przypadku, w którym poprawnie spełnia ona swoje zadanie (cień nie pada na płaszczyznę rzutowania) oraz wersji *depth fail* tam, gdzie zawodzi. Ogranicza się w ten sposób ilość wypełnianych wielokątów (*fillrate*) ponieważ domknięcia przednie oraz tylne brył nie są w takim przypadku wymagane i równocześnie podczas rysowania pomijane są niewidoczne fragmenty ich geometrii (odrzucone przez test głębokości). Tym samym obciążenie karty graficznej jest o wiele mniejsze. Wiąże się to niestety z dodatkowym zapotrzebowaniem na moc obliczeniową głównego procesora komputera, co jest związane z koniecznością podejmowania decyzji o wyborze trybu szablonu dla każdego z modeli. Niemniej jednak dla scen, w których obiekty mają wiele detali przybliżenie ich przy pomocy równoległościanu lub sfery, a następnie sprawdzenie czy taki obiekt znajduje się pomiędzy źródłem światła, a płaszczyzną rzutowania obrazu (rys. 17) może okazać się szybsze i przy tym równie bezbłędne niż stosowanie wyłącznie algorytmu *depth fail*.

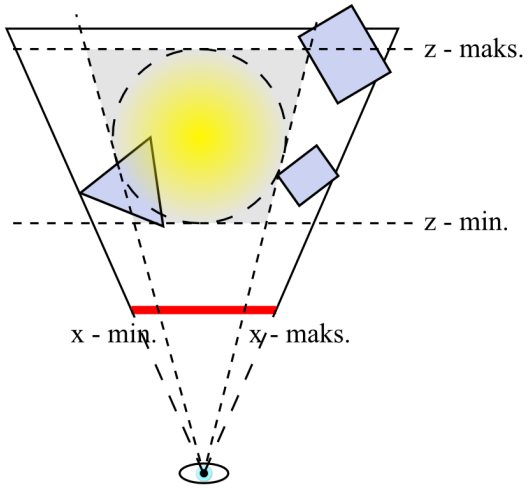


Rys. 17. Sposób wyboru typu operacji na szablonie. Jeżeli obiekt znajduje się wewnątrz ostrosłupa (czerwony trójkąt) o podstawie utworzonej z płaszczyzny rzutowania oraz wierzchołka w punkcie źródła światła konieczne jest użycie wersji *stencil fail*, zaś w pozostałych przypadkach można stosować algorytm *stencil pass*.

Jeżeli aplikacja korzysta ze źródła światła, którego wpływ na otoczenie zanika wraz z odległością można użyć oferowanego przez OpenGL testu nożyc (ang. *scissor test*) w celu ograniczenia obszaru rysowania cieni (technika szczegółowo opisana w [9] i [10]). Test nożyc pozwala ustawić prostokątny obszar obrazu poza, którym wpływ światła jest znikomy, a sama

scena jest pogrążona w cieniu (w konsekwencji zastosowania tylko samego modelu oświetlenia OpenGL). To udoskonalenie ponownie zmniejsza ilość wypełnianych wielokątów i tym samym obciążenie karty graficznej. W analogiczny sposób można ograniczyć zakres głębokości rysowania (ang. *z-bounds*) wykorzystując rozszerzenie `GL_EXT_depth_bounds`, o ile jest obsługiwane przez dany sterownik. Opisywane techniki ilustruje rysunek 18.

Najbardziej korzystną optymalizacją okazuje się zastosowanie programów wierzchołków (ang. *vertex programs*) opisanych w pozycjach [9] i [11]. Współczesne karty graficzne umożliwiają tworzenie małych programów wykonywanych przez kartę graficzną w momencie wykonywania transformacji punktów. Pozwala to m.in. zastąpić domyślne przekształcenia macierzowe w standardowym potoku przekształceń OpenGL (zob. dodatek C). Dzięki temu oprócz standardowych przekształceń można wykonać przy pomocy procesora karty graficznej transformację wierzchołków sylwetki obiektu oraz domknięcia tylnego czyli obliczyć równanie (36). Procesory graficzne z założenia są przystosowane do obliczeń macierzowych oraz wektorowych dzięki czemu znacznie lepiej radzą sobie z tego typu zadaniami. Tym samym generowanie cieni w mniejszym stopniu obciąża główny procesor komputera, a wypracowany w ten sposób zysk można wykorzystać do innych zadań.



Rys. 18. Ograniczenie obszaru rysowania brył cieni dla zanikającego źródła światła poprzez wykorzystanie testu nożyc (granice wartości współrzędnych x oraz y – niewidocznej na rysunku określonych we współrzędnych ekranowych) oraz ograniczenia zakresu głębi (współrzędnej z).

4. Mapowanie cieni (shadow mapping)

Działanie wszystkich z dotychczas opisywanych technik generowania cieni bazowało na przekształceniach geometrii. Metoda rzutowania geometrii wykorzystywała w tym celu odpowiednią macierz projekcji, zaś metoda cieni objętościowych tworzyła bryły zacienionych obszarów na podstawie obiektów sceny. Będąca przedmiotem rozważań niniejszego rozdziału technika mapowania cieni (ang. *shadow mapping*) prezentuje nieco inne podejście. Jej zastosowanie nie wymaga dokonywania skomplikowanych przekształceń wierzchołków modeli w celu uformowania obiektów, które bezpośrednio lub pośrednio mają posłużyć narysowaniu cieni. Algorytm wykorzystuje natomiast specjalną teksturę głębokości rzutowaną na modele sceny w celu późniejszego określenia zacienionych pikseli. Teksturę tę tworzy się dokonując renderingu sceny widocznej z kamery umieszczonej w pozycji źródła światła. Wartości kolorów tak utworzonego obrazu nie są w tej teksturze zapisywane, a jedynie zawartość z-bufora. Następnie powyższą teksturę rzutuje się na obiekty sceny w taki sposób jakby światło było projektorem. Przypomina to sytuację gdy w sali kinowej jakaś osoba znajdzie się wewnątrz snopa światła wychodzącego z projektora co sprawia, że fragment klatki filmu zostanie wyświetlony na jej ciele. Podczas renderingu rolę takiej osoby pełnią obiekty sceny, przy czym należy zaznaczyć, że inaczej niż w rzeczywistości obraz z powyższej projekcji nie jest blokowany przez żaden z nich. Oznacza to, że przenika on modele i pojawia się zarówno ze strony skierowanej w stronę światła jak i również przeciwnej. Ponadto wspomniana tekstura nie jest widoczna na renderingu ponieważ jak zostało już wcześniej wspomniane nie zawiera wartości kolorów, a jedynie zawartość bufora głębokości. Służy ona jedynie porównaniu zapisanej w niej odległości pierwszego widocznego z pozycji źródła światła obiektu z dystansem jaki dzieli światło od analizowanego fragmentu obiektu, na którym została „wyświetlona” ta tekstura. Dzięki temu można określić czy jest on oświetlony, czyli jest pierwszym napotkanym przez promień światła fragmentem jakiegoś obiektu, lub przeciwnie – jest w cieniu czyli znajduje się za blokującym światło fragmentem innego lub tego samego obiektu. Podsumowując jedynym działaniem na geometrii jakiego dokonuje opisywana metoda jest przekształcenie pozycji obserwatora do pozycji źródła światła w pierwszym przebiegu rysowania mającym na celu utworzenie tekstury głębokości. Dzięki temu w podstawowym wariancie powyższa technika

jest stosunkowo łatwa w implementacji, co przekłada się bezpośrednio na fakt, że jest obecnie najczęściej stosowaną metodą generowania cieni w grafice 3D czasu rzeczywistego. Dodatkowo aktualnie przeżywa ona swój intensywny rozwój m.in. dzięki wsparciu producentów sprzętu oraz zwiększających się możliwościach programowania procesorów graficznych.

4.1. Przebieg rysowania

Wykonanie pełnego renderingu sceny z wykorzystaniem metody mapowania cieni wymaga wykonania trzech przebiegów rysowania (zob. rys. 19). Pierwszy przebieg ma za zadanie wygenerować mapę cieni poprzez skopiowanie zawartości z-bufora utworzonej w wyniku renderingu sceny z kamery umieszczonej w pozycji źródła światła. Natomiast drugi etap przyczynia się częściowo już do powstawania właściwego obrazu. Jego zadaniem jest narysowanie całej sceny w cieniu czyli oświetlonej jedynie światłem otaczającym (ang. *ambient*). Następnie w trzecim przebiegu renderingu wykorzystywana jest tekstura głębokości przygotowana w pierwszej fazie algorytmu. Służy ona określeniu oświetlonych obszarów, które następnie są rozjaśniane poprzez dodanie światła rozproszonego (ang *diffuse*) i odbitego (ang. *specular*).

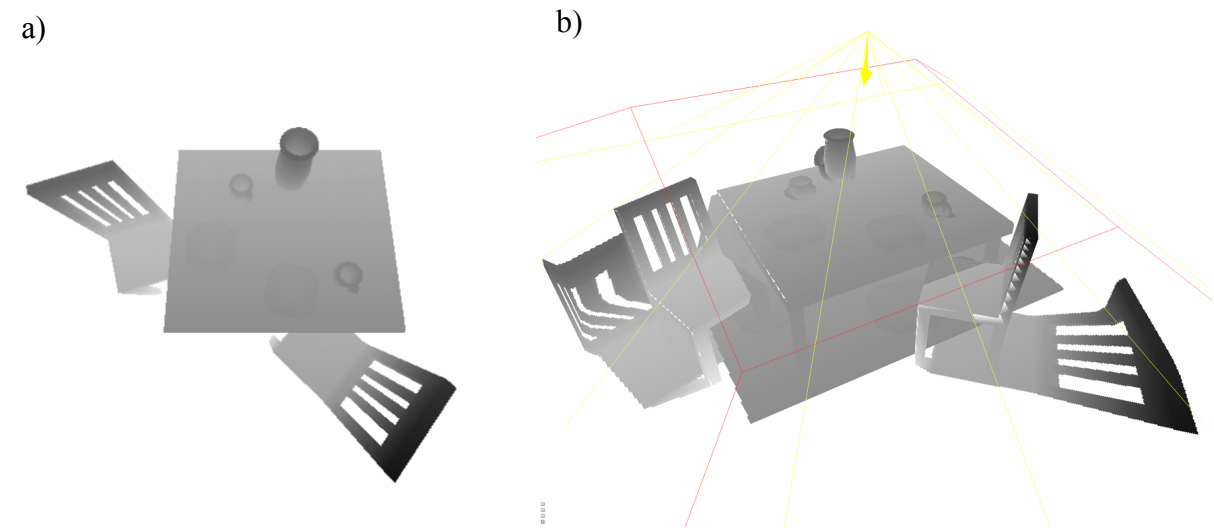
W pierwszym przebiegu rysowania należy wykonać wymienione poniżej czynności.

1. Ustawienie odpowiedniej macierzy projekcji.
2. Przemieszczenie kamery do pozycji źródła światła oraz skierowanie jej zgodnie z kierunkiem padania promieni.
3. Ustawienie parametrów widoku (okna).
4. Wyłączenie materiałów, oświetlenia, cieniowania oraz wszystkich innych niepotrzebnych efektów.
5. Wyłączenie zapisu do bufora kolorów.
6. Narysowanie sceny.
7. Skopiowanie zawartości bufora głębokości do tekstury.

Rys. 19. Diagram przedstawiający trzy przebiegi renderingu wymagane do wykonania obrazu przy pomocy metody mapowania cieni.

Pierwszym krokiem jest sprecyzowanie parametrów rzutowania wierzchołków. Macierz projekcji ustawiana w pierwszym punkcie musi mieć taką postać aby zdefiniowana przez nią frusta zawierała w swej objętości wszystkie obiekty sceny, które potencjalnie mogą rzucać cień lub go przyjmować. W przypadku reflektora może ona też być dodatkowo ograniczona do szerokości wytwarzanego przez niego snopa światła. Następnie należy umieścić obserwatora (kamerę) w pozycji źródła światła. Do wykonania powyższego przekształcenia należy wykorzystać kolejną macierz, przez którą mnoży się macierz model-widok. Obie macierze należy zapisać w celu późniejszego wykorzystania, gdyż będą one potrzebne w następnych etapach rysowania (zob. punkt 4.2). Ustawienie własności renderowanego obrazu dopełnia określenie parametrów widoku (ang. *viewport*) czyli pozycji oraz szerokości i wysokości (wyrażonych pikselach) tej części okna, w której ma być wykonany rendering. Tekstury posiadają pewne ograniczenia odnośnie swoich rozmiarów. Podstawowymi są maksymalna wysokość i szerokość oraz konieczność aby te wymiary były liczbą będącą potęgą 2. Tym samym dopuszczalne są wartości 1, 2, 4, 8, 16, 32, 64, 128, 256 itd. aż do maksymalnego rozmiaru obsługiwanego przez dany sterownik OpenGL. Niemniej jednak w najnowszych implementacjach (wspierających rozszerzenie `GL_ARB_texture_non_power_of_two` [13] lub w OpenGL 2.0 i nowszym [4]) ostatnia z tych restrykcji nie musi być przestrzegana i dozwolone są wszystkie wartości mieszczące się

w dozwolonym zakresie. Podsumowując szerokość i wysokość widoku musi odpowiadać żądanemu rozmiarowi tekstury oraz uwzględniać powyższe zastrzeżenia o ile takie występują. Ponadto wartości te muszą się mieścić w ograniczonym rozmiarze obszaru roboczego okna chyba, że został wykorzystany dodatkowy bufor pozaekranowy (ang. *offscreen buffer*). Taką możliwość musi jednak wspierać dana implementacja OpenGL. Jednym z takich rozwiązań jest rozszerzenie `GL_EXT_frame_buffer_object` (zob. [15]). Zanim przystąpi się do rysowania sceny warto wyłączyć wszelkie efekty mające wpływ na wizualną część renderingu ponieważ istotna w tym momencie jest tylko zawartość bufora głębokości, a nie sam wygląd obiektów. W takim razie można również zablokować zapis do bufora kolorów i tym samym zyskać na prędkości działania aplikacji. Powyższa czynność kończy realizację punktów 4 i 5. W tym momencie stan biblioteki OpenGL jest odpowiednio zainicjowany i można wykonać rendering obiektów po to, aby następnie skopiować powstałą w ten sposób zawartość z-bufora do tekstury. Oznacza to, że dla każdego piksela w tak utworzonej mapie przechowywana jest informacja o odległości pierwszego napotkanego przez promień światła fragmentu blokującego go obiektu (rys. 20a). Patrząc z pozycji źródła światła każdy punkt znajdujący się w odległości równej tej zapisanej w teksturze jest oświetlony, a każdy znajdujący się za nim jest w cieniu.



Rys. 20. Mapa cieni **a)** utworzona przy pomocy kamery ustawionej w punkcie źródła światła oraz **b)** jej projekcja na geometrię sceny.

Drugi przebieg rysowania jest najprostszą częścią opisywanego algorytmu. W rezultacie jego wykonania otrzymuje się scenę spowitą w mroku. Plan działania przedstawiony został poniżej.

1. Ustawienie macierzy projekcji oraz pozycji kamery.
2. Włączenie efektów oświetlenia, materiałów, pożądanej metody cieniowania i innych opcji wymaganych dla właściwego wyglądu obiektów.
3. Ustawienie parametrów światła.
4. Narysowanie sceny.

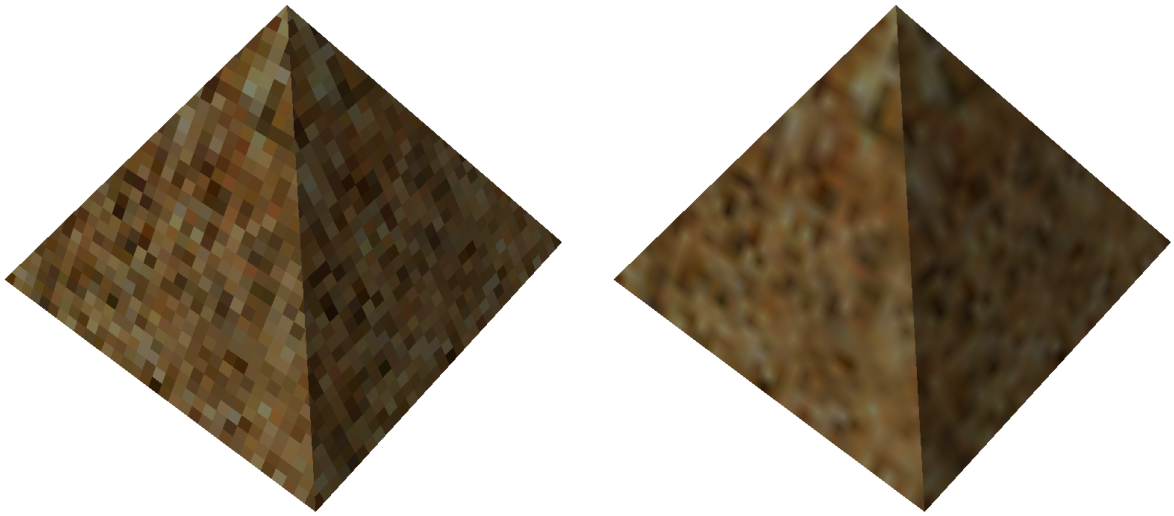
Pierwszy punkt ma za zadanie określić parametry rzutowania oraz umieścić kamerę we właściwej pozycji czyli takiej, z której obserwowana jest scena na końcowym obrazie. Następnie należy przywrócić wszelkie wykorzystywane w renderingu efekty. Kluczowy dla drugiego przebiegu rysowania jest punkt trzeci. Ponieważ na tym etapie cała scena ma znaleźć się w cieniu należy określić parametry oświetlenia. Podobnie jak miało to miejsce w przypadku metody cieni objętościowych te fragmenty sceny, do których nie dochodzi światło pozostają czarne lub oświetlone jedynie przez składową otaczającą modelu Phonga. W celu podkreślenia kształtu trójwymiarowych obiektów można również dodać odrobinę światła rozproszonego. Pozwoli to uwydatnić załamania na zacienionych fragmentach ich powierzchni. Po wprowadzeniu powyższych ustawień pozostaje jedynie wykonać odpowiednie funkcje rysujące i tym samym zakończyć drugi etap tworzenia obrazu.

Najbardziej złożoną częścią algorytmu jest wykonanie samych cieni. Zadanie to realizuje ostatni, trzeci przebieg renderingu. Szkic postępowania został przedstawiony poniżej.

1. Ustawienie parametrów filtrowania tekstury.
2. Ustawienie parametrów interakcji mapy.
3. Włączenie testu kanału alfa.
4. Wykonanie rzutowania tekstury.
5. Narysowanie sceny.

Ponieważ trzeci etap renderingu kontynuuje rysowanie zapoczątkowane w etapie drugim, a same parametry rzutowania oraz pozycja obserwatora nie ulegają zmianie to nie jest konieczne ponowne ustawianie macierzy projekcji oraz macierzy widoku. W takim razie pierwszą czynnością jest wybranie odpowiednich parametrów filtrowania tekstury czyli sposobu interpolacji pikseli podczas rozciągania (przybliżania) i pomniejszania (oddalania) nakładanej mapy w zależności od odległości od obserwatora. Dla zwykłej tekstury zazwyczaj używa się filtrowania dwuliniowego (ang. *bilinear filtering*), trójliniowego (ang. *trilinear filtering*) lub anizotropowego (ang. *anisotropic filtering*). Każda z tych metod pozwala

uzyskać gładkie przejścia pomiędzy tekselami (ang. *texel* od ang. *texture element*) czyli pikselami tekstury. Zapisany obraz bufora głębokości nie zawiera jednak wartości kolorów, a odległości obiektów widocznych z pozycji źródła światła od płaszczyzny rzutowania. Dlatego też nie można wygładzać przejść pomiędzy pikselami i należy zastosować najprostszą metodę filtrowania czyli interpolację najbliższego sąsiada (ang. *nearest-neighbor interpolation*). Polega ona na tym, że wybierana jest wartość teksela, którego środek znajduje się najbliżej analizowanego fragmentu co w przypadku powiększania standardowych obrazów uwidocznia poszczególne punkty mapy obrazkowej (efekt potocznie nazywany „pikselizacją" - zob. rys. 21).



Rys. 21. Sposoby filtrowania tekstury: z lewej interpolacja najbliższego sąsiada (ang. *nearest-neighbor interpolation*), a z prawej filtrowanie dwuliniowe (ang. *bilinear filtering*).

Kolejnym krokiem jest ustawienie sposobu interpretacji danej tekstury. Odwołując się ponownie do typowych zastosowań z wykorzystaniem zwykłego obrazu nakładanego na geometrię zazwyczaj ma on za zadanie określić kolory rysowanych prymitywów. To w jaki sposób zostanie to zrealizowane definiuje tryb środowiska tekstury (ang. *texture enviroment mode*), którym może być m.in. zastąpienie, dodanie, modulacja lub mieszanie koloru obrazu z pozostałymi kolorami obiektów. Dla przypadku cieni najbardziej odpowiedni będzie tryb modulacji czyli mnożenia ze sobą odpowiadających sobie komponentów kolorów danej tekstury C_T i koloru obiektu C_O.

$$C_M = (R_O \cdot R_T, G_O \cdot G_T, B_O \cdot B_T, A_O \cdot A_T) \tag{51}$$

W tym momencie rodzi się pytanie: skoro omawiana mapa nie posiada kolorów tylko wartości pochodzące z z-bufora to w jaki sposób można je łączyć z innymi kolorami? Otóż

dla tekstur głębokości istnieje pewien dodatkowy parametr określający sposób ich przekształcenia na wartości kolorów (GL_DEPTH_TEXTURE_MODE). Zapisaną w danym punkcie mapy liczbę można zinterpretować jako jasność (ang. *luminance*), intensywność (ang. *intensity*) lub wartość kanału alfa wynikowego koloru (zob. tab. 1).

Tab. 1. Możliwe wartości dla parametru trybu odwzorowania tekstury głębokości (GL_DEPTH_TEXTURE_MODE). Ostatnia kolumna tabeli przedstawia przekształcenie wartości z-bufora (D) w poszczególne składowe koloru: czerwoną (R), zieloną (G), niebieską (B) oraz alfa (A).

Wartość	**Znaczenie**	**Odwzorowanie w RGBA**
GL_LUMINANCE	Jasność koloru	(D, D, D, 1)
GL_INTENSITY	Intensywność koloru	(D, D, D, D)
GL_ALPHA	Wartość kanału alfa	(0, 0, 0, D)

W omawianym przykładzie wybrany jest tryb intensywności, co wraz z uprzednio ustawionym mnożeniem barwy obiektu przez wartość danego teksela pozwoli w rezultacie regulować jej jasność (i wartość kanału alfa). Takie rozświetlenie odbywać się będzie zgodnie z zapisaną w teksturze głębokością (D) czyli liczbą z przedziału $\langle 0;1 \rangle$. Nie jest to jednak w pełni zgodne z pożądanym zachowaniem. Zadaniem tekstury było bowiem określenie zacienionych obszarów poprzez porównanie tej wartości z odległością fragmentu obiektu od punktu źródła światła. W opisywanym przykładzie dystansowi temu odpowiada współrzędna r tekstury. Należy więc zweryfikować wynik nierówności (52) aby sprawdzić czy do danego punktu docierają bezpośrednio promienie danego światła.

$$r \leq D \tag{52}$$

Gdy powyższa relacja jest prawdziwa następuje rozjaśnienie koloru. Warto zauważyć, że warunek (52) można by teoretycznie zastąpić równością, gdyż mniejszość nigdy nie powinna wystąpić. Jednak w praktyce nierówność jest konieczna, gdyż nieuniknione są błędy numeryczne powstające w wyniku ograniczonej precyzji liczb (zwłaszcza jeżeli w grę wchodzi mnożenie). Sprawdzenie warunku (52) można wykonać poprzez ustawienie odpowiedniego trybu porównywania tekstury. Włącza się go ustawiając parametr tekstury GL_TEXTURE_COMPARE_MODE na wartość GL_COMPARE_R_TO_TEXTURE, zaś wyłącza stałą GL_NONE. Sposób porównywania określa parametr GL_TEXTURE_COMPARE_FUNC przyjmując te same wartości, co funkcja testu szablonu (por. dodatek D, tab. 2) i dla nierówności (52) będzie to GL_LEQUAL (mniejsze-równe). Dzięki powyższym ustawieniom

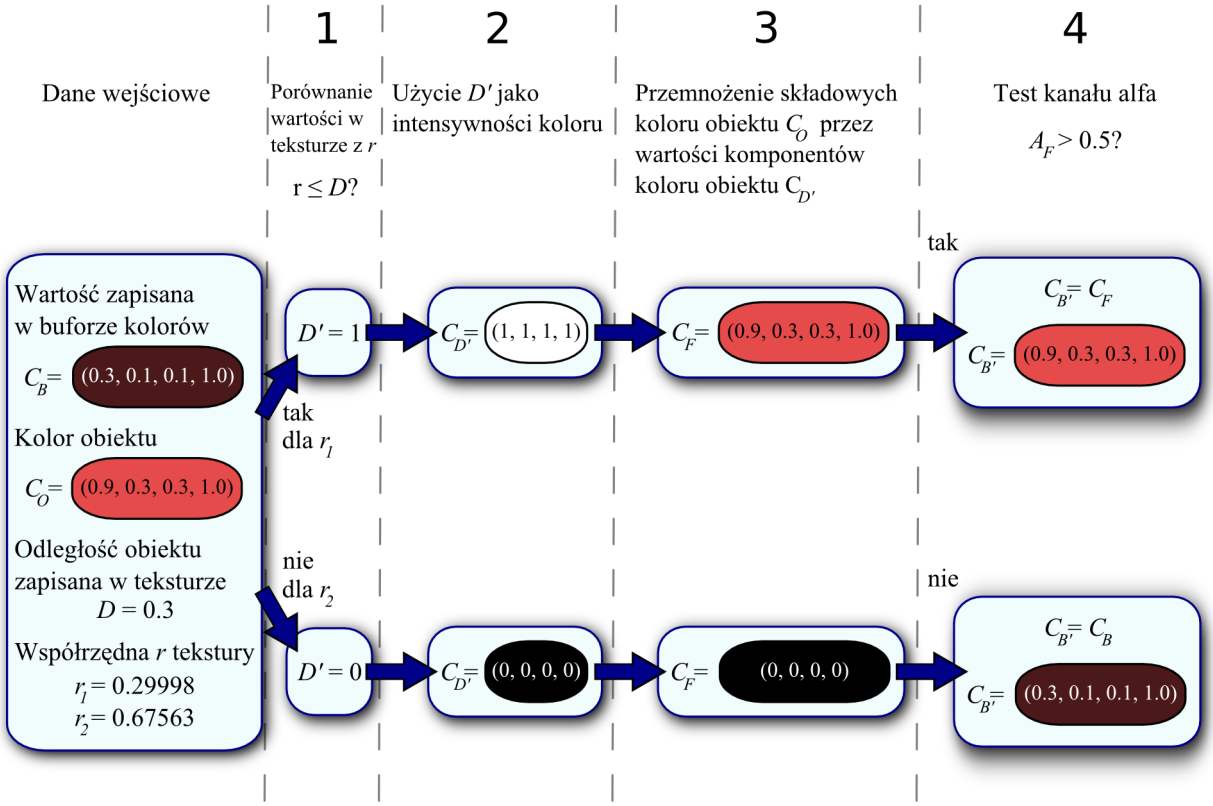
na obiekty nakładana jest mapa zer i jedynek co odpowiada kolorom RGBA (ponieważ wybrany jest tryb intensywności) równym (0,0,0,0) lub (1,1,1,1). Barwa obiektu zostaje przemnożona przez barwę teksela dając w rezultacie czerń lub kolor oryginalny. Utworzony w drugim przebiegu renderingu obraz zostaje całkowicie zastąpiony przez oświetlone i nieoświetlone (czarne) punkty. Traci się w ten sposób narysowane wcześniej cienie. Aby tego uniknąć należy skorzystać z testu kanału alfa (ang. *alpha test*), który nie pozwala nadpisać tych pikseli bufora kolorów, dla których przetwarzany fragment ma składową alfa (A) równą 0. Tym samym fragmenty mające w kanale alfa jedynkę (oświetlone) zastąpią piksele zapisane wcześniej w buforze ramki (cienie). Dla uniknięcia błędów numerycznych, które mogą wystąpić podczas mającego miejsce wcześniej mnożenia kolorów funkcja testu powinna być nierównością (np. (53)) odnoszącą się do wybranej z góry ustalonej liczby z przedziału $\langle 0\,;1\rangle$.

$$A > 0{,}5 \tag{53}$$

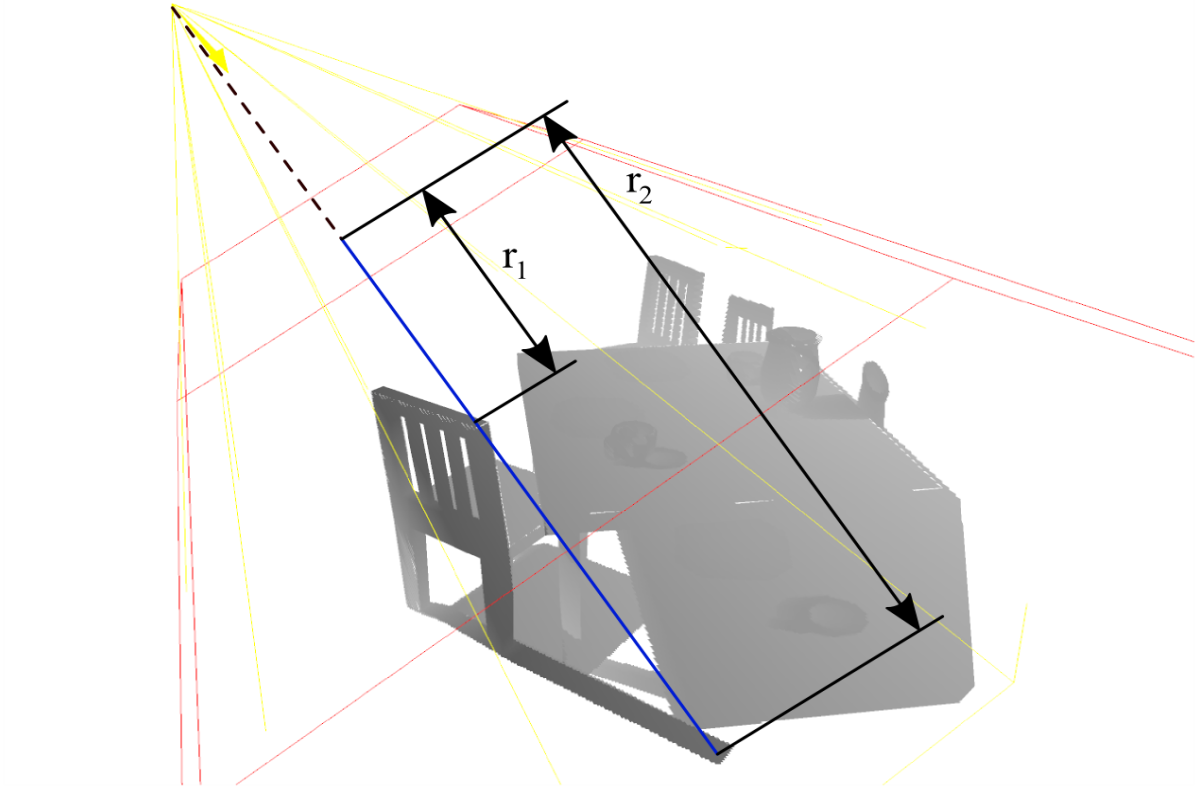
Wskazanie bibliotece OpenGL niniejszego zachowania testu kanału alfa kończy definiowanie ustawień przetwarzania tekstury. Punkt 4 przedstawionego wcześniej planu trzeciego przebiegu rysowania wskazuje, że następnym krokiem jest określenie odpowiednich współrzędnych tekstury. Definiują one w jaki sposób niniejszy obraz zostanie nałożony na geometrię. Jak zostało już wcześniej wspomniane tekstura musi zostać wyświetlona na obiektach w taki sposób jakby była obrazem wyświetlanym z projektora umieszczonego w pozycji źródła światła. Ponadto współrzędna r tekstury musi zostać prawidłowo określona tj. w taki sposób aby możliwe było prawidłowe określenie wyniku nierówności (52). Metoda realizacji powyższego zadania została dokładniej opisana w punkcie 4.2 niniejszej pracy. Ostatnią czynnością wykonywaną w ramach trzeciego przebiegu renderingu jest wykonanie odpowiednich funkcji rysujących obiekty sceny.

Powyższy opis etapu doświetlania odpowiednich obszarów może się wydawać dość skomplikowany dlatego został zaprezentowany także na diagramie z rysunku 22. Jego analiza będzie czytelniejsza jeżeli przeprowadzi się ją osobno dla każdego z przedstawionych przypadków r_1 oraz r_2. Na początku współrzędna tekstury $r = r_1 = 0{,}29998$ porównywana jest z wartością zapisaną w teksturze głębokości $D = 0{,}3$. Nierówność (52) jest spełniona w wyniku czego otrzymuje się nową liczbę $D' = 1$. Zgodnie z wcześniej ustawionym trybem interpretacji tekstury głębokości, w którym liczba D' określa intensywność barwy otrzymany zostaje kolor $C_{D'}$, który w tym przypadku odpowiada idealnej bieli (zob. tab.1).

Wynikowy kolor mnożony jest przez kolor oświetlonego obiektu C_O. Ponieważ ma miejsce mnożenie przez $C_{D'}=1$ to składowe barwy modelu nie ulegają zmianie. Wynik powyższego działania określa kolor fragmentu C_F. Ostatnim etapem jest wykonanie testu kanału alfa. Dzięki niemu aktualizacji podlegają tylko te punkty obrazu, których komponent alfa koloru jest równy 1 (a dokładniej większy od 0,5). Dla omawianego przykładu $r=r_1$ niniejszy warunek jest spełniony i kolor piksela w buforze C_B (barwa obiektu w cieniu) zastępowany jest kolorem fragmentu C_F. Tym samym przetwarzany fragment obiektu zostaje oświetlony. W drugim przypadku, gdzie współrzędna tekstury $r=r_2=0{,}67563$, warunek (52) nie jest prawdziwy. W efekcie otrzymuje się liczbę $D'=0$. Określa ona intensywność koloru $C_{D'}$, który jest idealną czernią z kanałem alfa równym 0. Przemnażając wartość jego składowych przez wartości składowych koloru obiektu C_O otrzymuje się ostateczny kolor fragmentu C_F będący również idealną czernią z komponentem alfa o wartości 0. O akceptacji niniejszej barwy piksela decyduje test kanału alfa. Daje on wynik negatywny, zatem przetwarzany fragment zostaje odrzucony. W buforze koloru pozostaje więc piksel przedstawiający obiekt w cieniu.



Rys. 22. Schemat przetwarzania fragmentu dla dwóch różnych przypadków wartości współrzędnej tekstury r_1 oraz r_2.



Rys. 23. Graficzna interpretacja współrzędnej *r* tekstury głębokości. Odcień szarości rzutowanej tekstury odpowiada odległości pierwszego napotkanego przez światło fragmentowi powierzchni obiektu. Zaznaczone zostały dwa opisywane przypadki r_1 oraz r_2 wartości współrzędnej tekstury. Czerwona linia określa krawędzie frusty renderingu mapy głębokości.

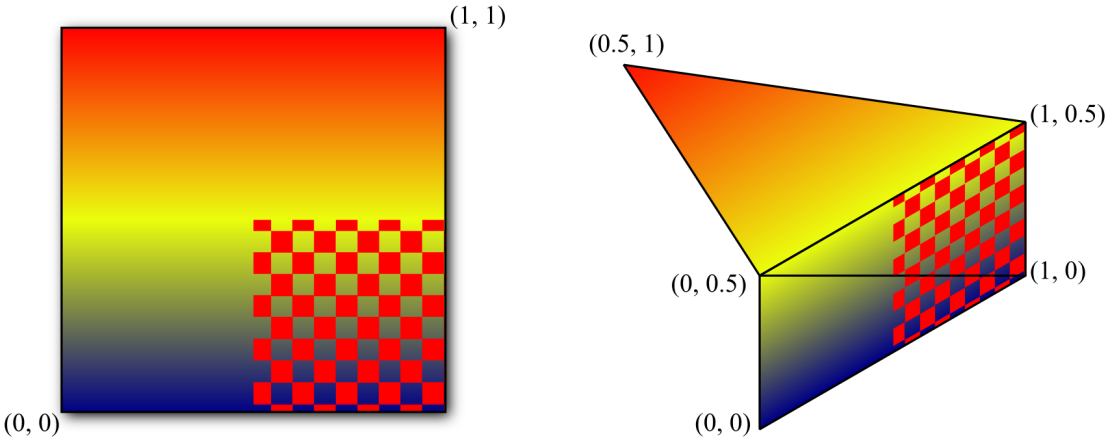
4.2. Współrzędne tekstury

Generowanie obrazu gdzie wszystkie obiekty mają powierzchnie o jednolitej barwie cieniowane jedynie w ramach modelu oświetlenia często nie pozwala uzyskać wystarczająco realistycznego efektu. W celu wzbogacania wyglądu renderowanych scen wykorzystuje się więc tekstury. Są one z reguły jedynie płaskimi obrazami (z wyjątkiem specjalnych tekstur trójwymiarowych), które nakłada się na geometrię modeli. W jaki sposób zostanie to zrealizowane określają koordynaty tekstury.

W przypadku dwuwymiarowego obrazu lewy, dolny róg tekstury ma współrzędne równe $(0,0)$ zaś prawy, górny $(1,1)$ (rys. 24). Programista może określić dla każdego wierzchołka odpowiednią parę wartości znajdującą się w przedziale $\langle 0,1 \rangle$, a nawet liczbę wykraczającą poza niego. Co się dzieje w takim przypadku wskazuje parametr zawijania tekstury (ang. *wrap*). Obraz poza swoimi granicami może być m.in. obcinany, powtarzany,

powtarzany z odbiciem albo po prostu mieć stałą barwę. Najbardziej adekwatną opcją dla omawianego algorytmu mapowania cieni jest wykorzystanie przycinania. Jest to logiczne rozwiązanie, gdyż nakładany obraz został wygenerowany dla pewnego ograniczonego obszaru sceny. Dlatego powtarzanie go, co jest ustawieniem domyślnym nie ma sensu i mogłoby powodować błędy. Podobnie jak w przypadku wierzchołków koordynaty tekstury można zdefiniować poprzez czterowymiarowy wektor (54).

$$\vec{T}=(s,t,r,q) \tag{54}$$

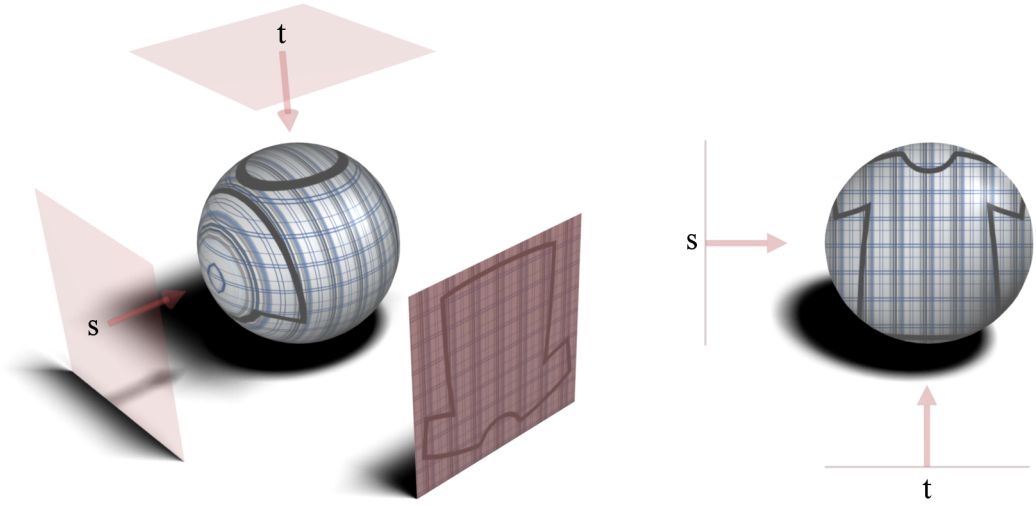


Rys. 24. Lewy rysunek przedstawia przykładową teksturę wraz zaznaczonymi koordynatami jej lewego-górnego i prawego-dolnego rogu. Z prawej strony zaprezentowany został fragment geometrii z nałożonym już obrazem. Liczby w nawiasach odpowiadają współrzędnym tekstury określonym dla danego wierzchołka.

Trzecia współrzędna ma zastosowanie dla trójwymiarowych tekstur i zazwyczaj nie wpływa na dwuwymiarowe obrazy. W omawianym przypadku posłuży ona jednak do określenia odległości dzielącej płaszczyznę rzutowania z pierwszego przebiegu renderingu i punkt, na którym wykonywana została projekcja analizowanego teksela tekstury (zob. rys. 23). Należałoby jeszcze zaznaczyć, że powyższy wektor, podobnie jak ma to miejsce w przypadku współrzędnych wierzchołków, przedstawiony jest we współrzędnych jednorodnych, a liczba q jest współczynnikiem skalowania pozostałej trójki.

Ręczne definiowanie współrzędnych teksturowania dla każdego wierzchołka z osobna nie jest jedyną możliwością jaką oferuje biblioteka OpenGL. Potrafi ona również wyręczyć programistę i samodzielnie wygenerować odpowiednie współrzędne zgodnie z określonymi przez niego wytycznymi. Jest to szczególnie ważne w przypadku bardziej skomplikowanych zastosowań, w których byłyby wymagane dość złożone obliczenia dodatkowo obciążające główny procesor komputera (CPU). Można więc skorzystać z kilku predefinowanych trybów generowania współrzędnych tekstury. Najbardziej istotne z punktu widzenia niniejszej pracy

są współrzędne liniowe względem obiektu (ang. *object linear*) lub względem oka (ang. *eye linear*) nazwane również planarnymi (płaskimi). W obu przypadkach dla każdej ze współrzędnych tekstury s, t, r oraz q określa się osobno płaszczyznę, której wektor normalny wskazuje jej kierunek (rys. 25).



Rys. 25. Rezultat automatycznego generowania liniowych koordynatów tekstur we współrzędnych obiektu (z lewej) oraz współrzędnych oka (z prawej). Aby uzyskać powyższy efekt czerwone powierzchnie powinny być styczne do teksturowanych kul jednak w celu pokazania wektorów normalnych zostały one odsunięte.

Różnica pomiędzy dwoma omawianymi wariantami polega na tym, że w pierwszym przypadku owa płaszczyzna określona jest w układzie współrzędnych obiektu, zaś w drugim w układzie współrzędnych oka (czyli po zastosowaniu macierzy model-widok). Zgodnie z dokumentacją biblioteki OpenGL [4] wygenerowanie pojedynczej koordynaty (s, t, r lub q) dla trybu liniowego względem obiektu polega na wyznaczeniu iloczynu skalarnego wektora położenia wierzchołka $\vec{V}$ (we współrzędnych obiektu) z wektorem normalnym płaszczyzny $\vec{P}$ (wyrażonym również we współrzędnych obiektu).

$$g = V_{x_o} \cdot P_x + V_{y_o} \cdot P_y + V_{z_o} \cdot P_z + V_{w_o} \cdot P_w \tag{55}$$

Obliczenia w przypadku drugiej metody są podobne tylko, że w tym wariancie współczynniki płaszczyzny określone są we współrzędnych oka:

$$g = V_{x_e} \cdot P'_x + V_{y_e} \cdot P'_y + V_{z_e} \cdot P'_z + V_{w_e} \cdot P'_w, \tag{56}$$

gdzie wektor płaszczyzny $\vec{P}'$ jest przetransformowanym do układu współrzędnych obiektu wektorem płaszczyzny:

$$P_o = (P_{x_e}, P_{y_e}, P_{z_e}, P_{w_e}) M_{MV}^{-1} = (P'_x, P'_y, P'_z, P'_w). \tag{57}$$

Wynik każdego z powyższych iloczynów skalarnych jest odległością analizowanego wierzchołka od danej powierzchni i tym samym określa wartość żądanej koordynaty.

Transformacja (57) jest konieczna, ponieważ w momencie definiowania płaszczyzny obiekt mógł zostać poddany działaniu macierzy model-widok, a z krótkiej analizy rysunku 25 można wywnioskować, że mapowanie tekstury w trybie liniowym względem oka jest niezależne od przekształceń wykonywanych na obiekcie.

Projekcja obrazu wykonanego podczas pierwszego przebiegu renderingu będzie częściowo opierała się na wygenerowanych współrzędnych planarnych względem oka. Automatyczne wyliczenie koordynatów pozwoli nałożyć mapę głębokości tak jakby znajdowała się w płaszczyźnie rzutowania frusty (prawa kula na rys. 25). Powyższa tekstura została jednak wyrenderowana z punktu widzenia światła, a nie z pozycji kamery (oka) dlatego też należy wykonać odpowiednią transformację wyliczonych w ten sposób współrzędnych tekstury. Biblioteka OpenGL dysponuje odpowiednim do tego celu mechanizmem jakim jest macierz tekstury. Wierzchołki obiektów rysowanych w pierwszym przebiegu renderingu poddane były przekształceniom modelu, widoku oraz projekcji (58).

$$M = M_P \cdot M_V \cdot M_M \tag{58}$$

Ważną czynnością podczas rysowania tekstury było zapamiętanie wykorzystywanej macierzy projekcji M_P oraz macierzy model-widok zawierającej (w momencie zapisu) jedynie M_V z równania (58) i tym samym określającej transformację pozycji obserwatora. Poszczególne przekształcenia obiektów M_M dokonywane były już po zapisaniu powyższych struktur. Wykorzystując te informacje można wykonać rzutowanie obrazu poprzez wykorzystanie wspomnianej macierzy tekstur. Wystarczy więc powtórzyć opisane przekształcenia, aby przybliżyć się do zamierzonego efektu. Pozostaje jeszcze jeden problem jakim jest fakt, że otrzymana w ten sposób wartość współrzędnej r we współrzędnych przycięcia zawierać się będzie w zakresie $\langle -1,1 \rangle$ natomiast wartości D w teksturze w przedziale $\langle 0,1 \rangle$. Rozwiązaniem jest dopasowanie wartości r tak aby zachodziła pomiędzy nimi zgodność. W tym celu należy ją przeskalować ze współczynnikiem $0{,}5$ oraz przesunąć (odchylić, ang. *bias*) również o liczbę $0{,}5$. Ostatecznie macierz tekstury M_T wylicza się w następujący sposób (59).

$$M_T = M_B \cdot M_S \cdot M_P \cdot M_V \tag{59}$$

Dzięki powyższym zabiegom można z powodzeniem sprawdzić wynik nierówności (52) i co się z tym wiąże określić zacienione obszary.

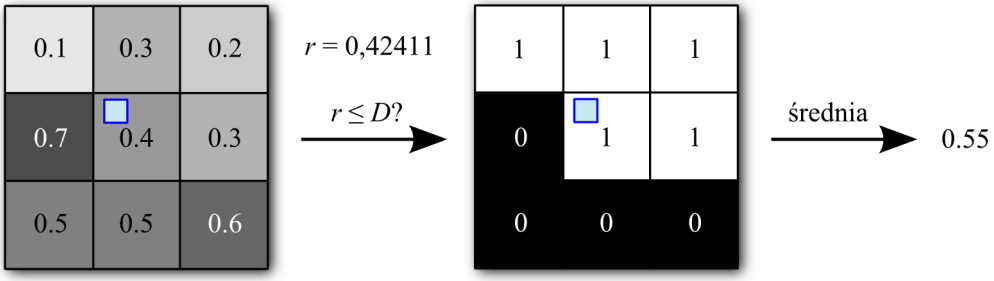
4.3. Udoskonalenia

W ostatnich latach technika mapowania cieni przy pomocy tekstur głębokości przechodzi swój gwałtowny rozwój. Algorytmy ulepszające podstawową zasadę jej działania znane są już od kilkunastu lat ale dopiero w ostatnim czasie pojawiły się rozwiązania sprzętowe umożliwiające ich wydajną implementację w grafice czasu rzeczywistego. W tej chwili każdy nowy komputer do zastosowań domowych może tworzyć płynne animacje złożone z obrazów o jakości, która parę lat temu była dostępna jedynie dla wyspecjalizowanych stacji graficznych.

W 2002 roku zostało zatwierdzone rozszerzenie `GL_ARB_shadow_ambient` dedykowane technice mapowania cieni, które pozwala na zmniejszenie ilości przebiegów renderingu z trzech do dwóch (zob. [2]). Dzięki niemu możliwe jest wykonanie jednoczesnego rysowania sceny oświetlonej światłem otaczającym i sceny oświetlonej światłem rozproszonym oraz odbitym (obiekty nie zacienione). Niestety pomimo kilku lat, które minęły od jego wprowadzenia jedynie firma ATI wspiera powyższe rozwiązanie. Jednak w dobie w pełni programowalnych układów graficznych ten sam efekt można uzyskać stosując programy fragmentów (ang. *fragment programs* lub *pixel shaders*).

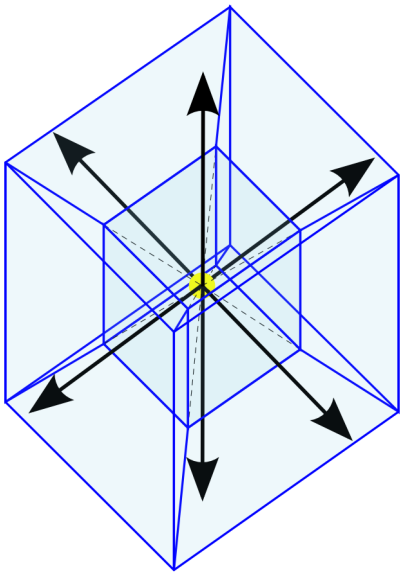
Zastosowanie programów fragmentów nie ogranicza się jedynie do przyspieszenia algorytmu, ale może posłużyć również poprawieniu jakości generowanego obrazu. Jednym z najbardziej zauważalnych problemów jest ograniczony rozmiar tekstury. Przy dużych zbliżeniach lub zbytnim rozciągnięciu mapy na powierzchni obiektów uwidaczniają się jej pojedyncze piksele. Jak zostało już wcześniej wspomniane w punkcie 4.1 ponieważ tekstura zawiera informacje o głębi, a nie o kolorach nie można stosować wygładzania dwuliniowego. Interpolacji nie można wykonać przed porównaniem wartości głębokości zapisanej w teksturze z wartością współrzędnej r tekstury natomiast można interpolować rezultaty tego działania (52). Jedną z takich technik jest *Percentage Closer Filtering* (zob. [11] i [15]), które pozwala uśrednić wartości kanału alfa danego teksela biorąc pod uwagę wartości jego sąsiadów (rys. 26). Zwiększenie ilości próbek do obszarów o rozmiarze nawet 16x16 daje bardzo dobre efekty. Metodę tę można dodatkowo jeszcze wzbogacić o filtrowanie dwuliniowe (interpolację) wartości binarnych zamiast obliczania zwykłej średniej powstałych zer i jedynek. Dzięki temu fragmenty znajdujące się w obrębie tego samego teksela będą miały różne wartości w zależności od ich położenia (a nie jedną tę samą obliczoną ze

średniej). Prezentacja [15] proponuje kolejne ulepszenia polegające m.in. na niejednorodnym pobieraniu sąsiadów co sprawia, że znika efekt mozaiki („pikselizacji"). Niestety zwiększenie jakości kosztuje sporo mocy obliczeniowej karty graficznej dlatego można skorzystać z dodatkowej mapy zawierającej sylwetki obiektów widocznych z punktu położenia źródła światła. Taka tekstura pełni rolę maski, która rzutowana na geometrię sceny razem z mapą głębokości pozwala stosować dokładniejsze filtrowanie PCF tylko na krawędziach cieni.



Rys. 26. *Percentage Closer Filtering* dla obszaru wielkości 3x3 teksele. Z lewej strony zostały pokazane oryginalne wartości tekstury. Każdy fragment (niebieski kwadrat) znajdujący się wewnątrz środkowego teksela będzie miał przypisaną wartość kanału alfa, która powstała z uśrednienia wyników porównywania wartości *r* dla tego fragmentu z głębokościami zapisanymi w jego tekselu oraz sąsiadach jego teksela.

Jedną z wielu trudności przy implementacji techniki mapowania cieni jest ograniczanie jej zastosowania do świateł kierunkowych oraz reflektorów. Biblioteka OpenGL nie pozwala na tworzenie obrazów gdzie kąt pomiędzy lewą i prawą płaszczyzny frusty jest równy lub większy niż 180 stopni. Tym samym nie jest możliwe wygenerowanie mapy głębokości dla świateł punktowych. Rozwiązaniem okazuje się wykorzystanie dostępnej w OpenGL tekstury kubicznej złożonej z sześciu zwykłych obrazów. Wiąże się to niestety z koniecznością wielokrotnego rysowania całej sceny podczas generowania map głębokości, jednorazowo dla każdego z kierunków (rys. 27). Obliczenie odpowiednich współrzędnych tekstury również nie jest trywialne i wymaga użycia programów wierzchołków oraz fragmentów (zob. [11]), ponieważ OpenGL w standardzie nie posiada odpowiedniej do tego zadania metody generowania współrzędnych.



Rys. 27. Sześć frust i kierunków rzutowania dla mapy kubicznej dla przypadku punktowego źródła światła.

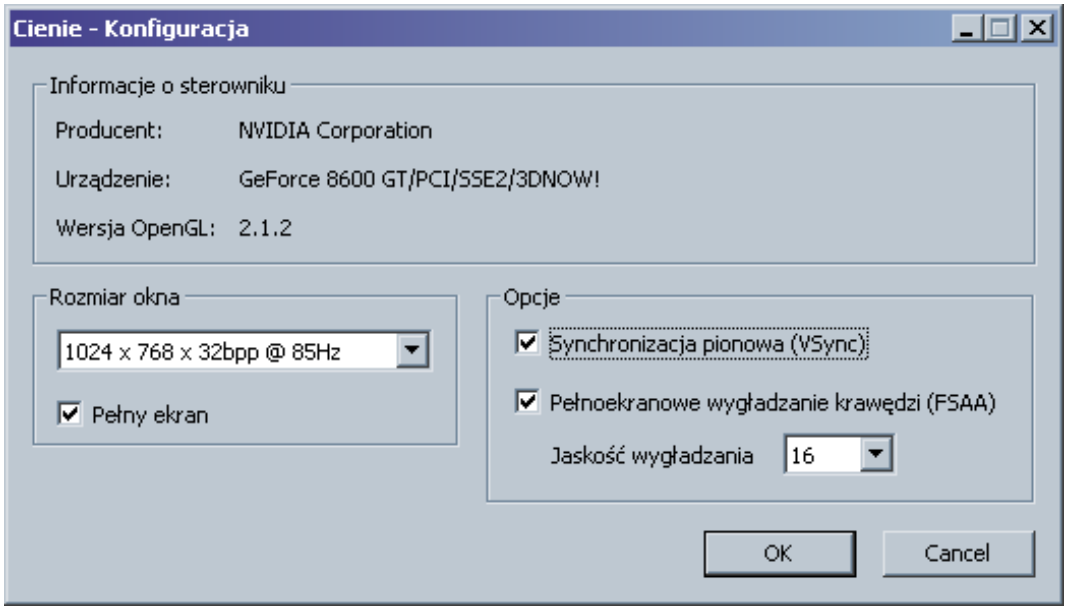
5. Aplikacja demonstracyjna

Częścią przygotowywanej przeze mnie pracy magisterskiej była samodzielna implementacja opisanych w poprzednich rozdziałach algorytmów generowania cieni. Przygotowana została również aplikacja demonstracyjna pozwalająca na testowanie owej implementacji dla różnych, dobieranych w trakcie działania programu, parametrów. Implementacja obejmuje zatem trzy techniki: rzutowanie cieni, cienie objętościowe oraz mapowanie cieni. Działanie programu nie ogranicza się tylko do prezentacji wyniku algorytmu, ale również pozwala na analizę sposobu jego działania, w tym słabości i zalet poszczególnych technik. Dla przykładu w przypadku cieni objętościowych widoczne mogą być bryły cieni, a w przypadku mapowania cieni, tekstura głębokości wyznaczająca miejsca zacienione.

5.1. Obsługa programu

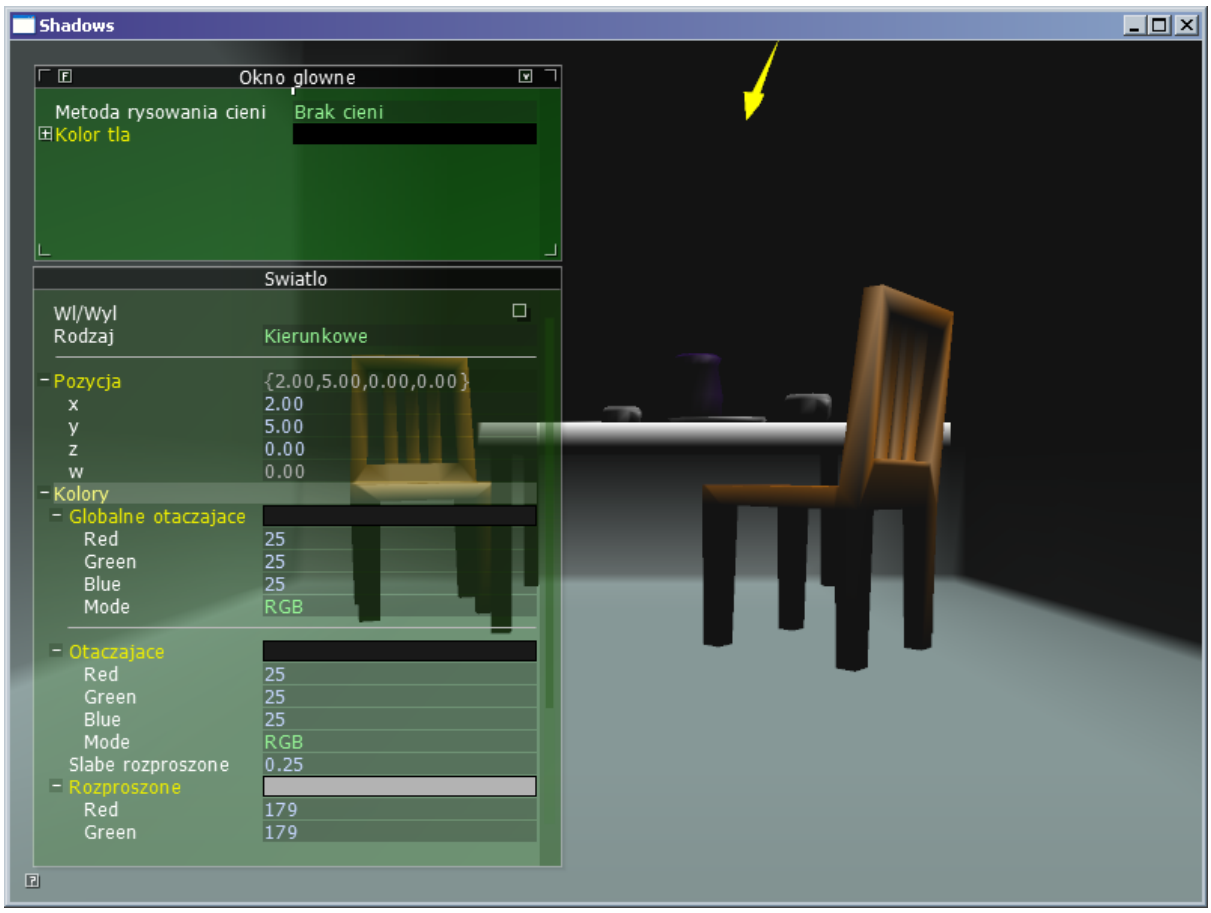
W momencie uruchomienia aplikacji jako pierwsze pojawia się okno konfiguracji wyświetlania (rys. 28). Pozwala ono sprawdzić producenta, nazwę oraz wersję sterownika OpenGL. Jednak przede wszystkim można dzięki niemu wybrać rozmiar okna, w którym wyświetlana będzie grafika 3D lub rozdzielczość ekranu, jeżeli do jej wyświetlania stosowany będzie tryb pełnoekranowy. Kolejną dostępną opcją jest możliwość włączenia synchronizacji pionowej obrazu. Gdy jest ona aktywna, kolejne klatki animacji są generowane w momencie odświeżenia obrazu monitora. Usunięcie zaznaczenia tej opcji powoduje, że obraz może być o wiele częściej aktualizowany, a tym samym karta graficzna nie musi czekać na kolejny cykl odświeżenia ekranu. Dla przykładu jeżeli monitor odświeżany jest częstotliwością rzędu 60Hz możliwe jest uzyskanie nawet 100 i więcej klatek animacji na sekundę. Oczywiście część z wygenerowanych klatek nie zostanie wyświetlona. Opcja ta jest dostępna tylko wtedy, gdy jest obsługiwana przez sterownik obecny w komputerze. Pełnoekranowe wygładzanie krawędzi (ang. *full screen antialiasing*) umożliwia polepszenie jakości obrazu poprzez utworzenie gładszych przejść na krawędziach obiektów. Jest to szczególnie przydatne w przypadku skośnych linii. Jakość wygładzania można regulować ale należy mieć na uwadze,

że zwiększenie ilości próbek skutkuje większym zapotrzebowaniem na pamięć oraz spowolnieniem wyświetlania. Z tego powodu w przypadku gdy wybrana rozdzielczość jest bardzo wysoka i towarzyszy temu wysoka jakość wygładzania, może zabraknąć pamięci na karcie graficznej i w efekcie program w ogóle się nie uruchomi. Pełnoekranowe wygładzanie krawędzi musi być obsługiwane przez sterownik i tym samym opcja ta może nie być niedostępna.



Rys. 28. Okno konfiguracji wyświetlania.

Po zatwierdzeniu przyciskiem „OK” konfiguracji wyświetlania pojawia się główne okno programu (rys. 29). Na początku widoczna jest oświetlona scena, pozbawiona cieni. W obrębie okna widoczne są również okna narzędziowe pozwalające na dostosowanie parametrów rysowania. Każde z tych okien można zminimalizować korzystając z przycisku znajdującego się w jego prawym górnym rogu. Po zminimalizowaniu okno znika, a w zamian w lewym dolnym rogu ekranu lub okna macierzystego pojawia się jego ikona. Jedna z takich ikon znajduje się we wskazanym miejscu zaraz po uruchomieniu aplikacji i dotyczy okna pomocy. Po jej naciśnięciu można dowiedzieć się jaką funkcję pełnią poszczególne kontrolki okien narzędziowych. Wyświetlana pomoc zawsze dotyczy tylko widocznych parametrów. Z tego powodu dopóki, niektóre z tych parametrów nie staną się dostępne, ich opis nie będzie widoczny. Z lewej strony belki tytułowej każdego okna narzędziowego znajduje się przycisk oznaczony dużą literą „F”, który umożliwia zmianę rozmiaru czcionki. Podobnie jak w przypadku standardowych okien systemu Windows, rozmiar okien narzędziowych można regulować „łapiąc” i przesuwając ich rogi.



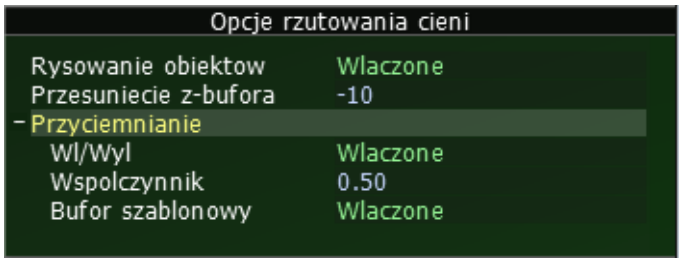
Rys. 29. Główne okno programu.

Główne okno narzędziowe programu (zob. rys. 29) umożliwia wybór odpowiedniej metody rysowania cieni oraz ustawienie koloru tła. Jeżeli z rozwijanej listy wybrana zostanie jedna z omówionych w tej pracy technik generowania cieni, to z prawej strony ekranu pojawi się kolejne okno narzędziowe pozwalające na dostosowanie jej parametrów. Drugie podstawowe okno narzędziowe aplikacji służy do kontrolowania oświetlenia. Podstawowymi funkcjami, jakie ono oferuje są: wybór rodzaju światła (punktowe, kierunkowe lub reflektor), ustawienie jego pozycji oraz kolorów. W przypadku zastosowania reflektora dostępne są dodatkowe parametry umożliwiające dostosowanie zmiennych stożka światła oraz ustalenie jego kierunku w przestrzeni. Światło w programie zaznaczone jest przez symbolizującą go żółtą, trójwymiarową ikonę (zob. rys. 29). Światło punktowe przedstawia mała sfera, kierunkowe – strzałka, natomiast reflektor – strzałka oraz stożek pokazujący jego zasięg. Przypomnę, że źródło światła kierunkowego znajduje się w nieskończoności (współrzędna $w=0$) natomiast jego ikona wyświetlana jest w punkcie wyznaczonym przez współrzędne x, y i z źródła światła, lecz z pominięciem współrzędnej w (czyli $w=1$). Każda ze współrzędnych i parametrów opisujących źródła światła można wprowadzić z klawiatury lub

skorzystać z o wiele szybszego w obsłudze kołowego kontrolera o nazwie RotoSlider (rys. 30). Jego aktywacja następuje, gdy użytkownik „przytrzyma” myszką nazwę parametru lub przycisk oznaczony kropką znajdujący się po prawej stronie przycisku ze znakiem plusa. Im krótsza jest czerwona kreska, która się wówczas pojawi, tym szybciej można zmieniać wartości. Natomiast jej wydłużenie pozwala na bardziej precyzyjną regulację wartości.

Rys. 30 . RotoSlider.

Metoda rzutowania cieni (w momencie gdy jest aktywna) może zostać dowolnie skonfigurowana przy pomocy dedykowanego jej okna narzędziowego (rys. 31). Wyłączenie rysowania obiektów umożliwia zobaczenie samych cieni, które są generowane w drugim przebiegu renderingu. W przypadku aktywowania tej opcji dobrze jest ustawić jakiś jasny kolor tła ponieważ bez niego cienie mogą być niewidoczne (czarny ekran). Przesunięcie z-bufora jest parametrem pozwalającym uniknąć błędów pokrywania (przenikania) się cieni z obiektem „przyjmującym” je (w scenie użytej w programie rolę tą pełni podłoga pokoju). Grupa opcji „Przyciemnianie” umożliwia włączenie półprzeźroczystych cieni. Dobór odpowiedniego współczynnika pozwala regulować ich jasność (0 – niewidoczne, 1 – w pełni czarne cienie). Włączenie bufora szablonowego zapobiega nakładaniu się na siebie cieni powstałych z różnych fragmentów obiektów.



Rys. 31. Okno parametrów rzutowania cieni.

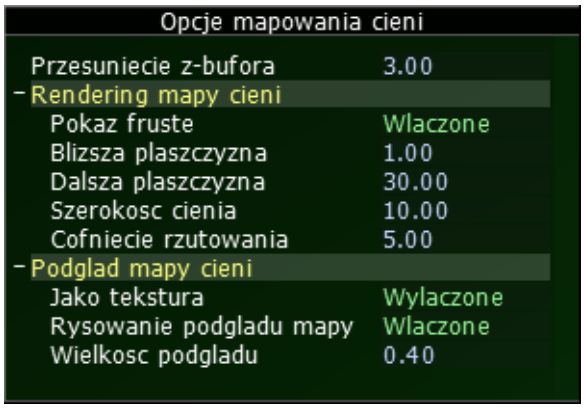
Przełączenie aplikacji w tryb rysowania z wykorzystaniem metody cieni objętościowych uaktywnia kolejne okno narzędziowe pozwalające na dostosowanie parametrów związanych z tą techniką (rys. 32). Podstawową dostępną wówczas opcją jest wybór typu operacji na szablonie: *depth pass* lub *depth fail* (zob. komentarz na ten temat w rozdziale 3.3). Następny parametr uaktywnia macierz projekcji OpenGL, w której tylna płaszczyzna obcinania znajduje się w nieskończonej odległości od obserwatora (zob.

punkt 3.4). Interesującą funkcją jest możliwość rysowania brył cieni. W momencie jej włączenia bryły te zostaną narysowane w postaci półprzeźroczystych, czerwonych obiektów. Grupa opcji „Rodzaj brył" umożliwia zastosowanie brył cieni o skończonej lub nieskończonej długości. W pierwszym przypadku można ustawić tę długość korzystając z odpowiedniej kontrolki. Szybkie przyciemnianie pozwala uniknąć drugiego przebiegu rysowania sceny mającego za zadanie rozjaśnienie niezacienionych obszarów (zob. rys. 7a) poprzez narysowanie półprzeźroczystego, czarnego prostokąta przyciemniającego miejsca, w których występują cienie (zob. rys. 7b). Podobnie jak to miało miejsce w omawianej w poprzednim akapicie metodzie rzutowania cieni, także tutaj można regulować jasność cieni.

Rys. 32. Okno opcji cieni objętościowych.

Okno opcji sterujących algorytmem mapowania cieni zostało przedstawione na rysunku 33. Przesunięcie z-bufora pozwala ustawić odchylenie wartości w buforze głębi podczas rysowania mapy cieni. Przy odpowiednim dobraniu wartości można uniknąć migotania i przenikania fragmentów cienia na powierzchniach obiektów. Wartość tego parametru zależy od dokładności bufora głębi, specyfiki danej karty graficznej oraz parametrów rzutowania (pozycji kamery, źródła światła, macierzy projekcji itp.). Grupa opcji o nazwie „Rendering mapy cieni" dostosowuje parametry rzutowania podczas tworzenia mapy głębokości. Można więc określić pozycje bliższej oraz dalszej płaszczyzny obcinania oraz w przypadku kierunkowego źródła światła, gdzie rzutowanie jest równoległe, szerokości frusty oraz cofnięcie kamery podczas wykonywania renderingu (ponieważ światło znajduje się w nieskończoności). Regulacja odległości bliższej płaszczyzny obcinania powinna być dokonywana ostrożnie ponieważ zbyt mała odległość może spowodować utratę precyzji bufora głębi. Natomiast zbyt duża wartość może sprawić, że część obiektów znajdzie się za jej tylną stroną czyli pomiędzy pozycją źródła światła, a przednią płaszczyzną obcinania. Dostosowanie opcji renderingu mapy cieni umożliwia włączenie wyświetlania frusty (czerwone linie), która pozwala na zobaczenie obszaru sceny objętego zasięgiem rysowania

podczas tworzenia mapy głębi. Grupa parametrów „Podgląd mapy cieni” umożliwia podejrzenie wygenerowanej tekstury głębokości. Można wyświetlić jej podgląd w prawym, dolnym rogu okna oraz zobaczyć, w jaki sposób jest rzutowana na obiekty sceny. Odległości w niej zapisane są wyświetlane w taki sposób, że bliższe obiekty są ciemniejsze, natomiast dalsze – jaśniejsze. Metoda mapowania cieni jest dostępna wyłącznie jeżeli sterownik obsługuje OpenGL w wersji 1.4 lub wyższej. W przypadku gdy to wymaganie nie będzie spełnione na wygenerowanym obrazie pojawi się jedynie trójwymiarowa ikona światła.



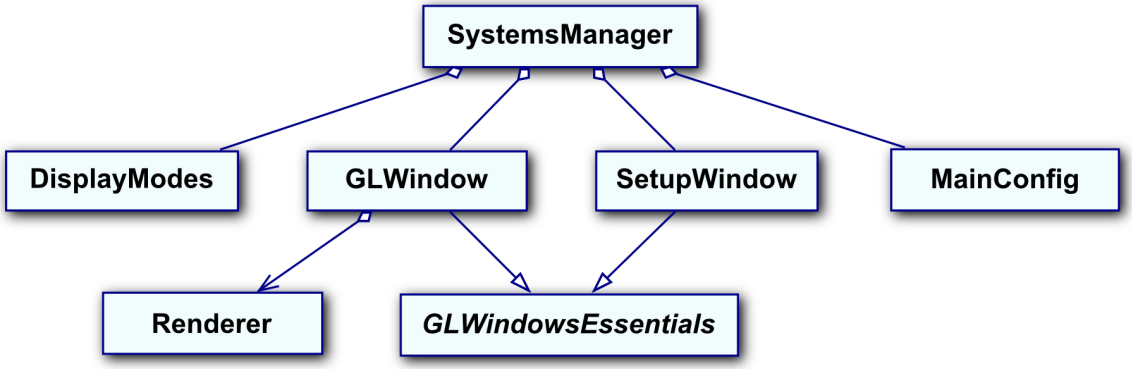
Rys. 33. Okno parametrów metody mapowania cieni.

Poza parametrami dostępnymi z poziomu okien narzędziowych możliwe jest również sterowanie kamerą przy pomocy myszy. Przytrzymanie jej lewego przycisku i następnie poruszenie nią obraca kamerę wokół środka układu współrzędnych. Przytrzymanie środkowego (trzeciego) przycisku i poruszanie muszą w górę i w dół pozwala przybliżać i oddalać widok. Z kolei przytrzymanie prawego przycisku powoduje przesuwaniem kamery w płaszczyźnie ekranu (lewo, prawo, góra, dół). Warto również odnotować, że w przypadku uruchomienia aplikacji w trybie pełnego ekranu zamknięcie jej może nastąpić po naciśnięciu standardowego skrótu klawiaturowego Windows tj. Alt+F4.

5.2. Szczegóły implementacji

Kod programu został napisany w języku C++, a sama aplikacja jest podzielona na moduły. Każdy z nich odpowiada za pewną część przydzielonych mu zadań dzięki czemu możliwe jest spełnienie jednego z głównych założeń programowania obiektowego, czyli hermetyzacji, co jest generalnie dość trudne do uzyskania w przypadku programów korzystających z grafiki trójwymiarowej. Struktura klas (rys. 34) jest uproszczonym

zastosowaniem wzorca projektowego zaproponowanego w artykule [16]. Na szczycie hierarchii stoi menadżer systemów `SystemsManager`, który odpowiada za tworzenie i zarządzanie głównymi modułami programu. Przechowuje on również wszystkie współdzielone przez moduły klasy oraz podstawowe dane programu (np. uchwyt aplikacji, ścieżkę do jej katalogu itd.). Każdy z systemów posiada wskaźnik do zarządcy i tym samym może poprzez niego komunikować się z pozostałymi modułami (dla czytelności diagramu z rysunku 34 relacje pomiędzy modułami nie zostały pokazane).



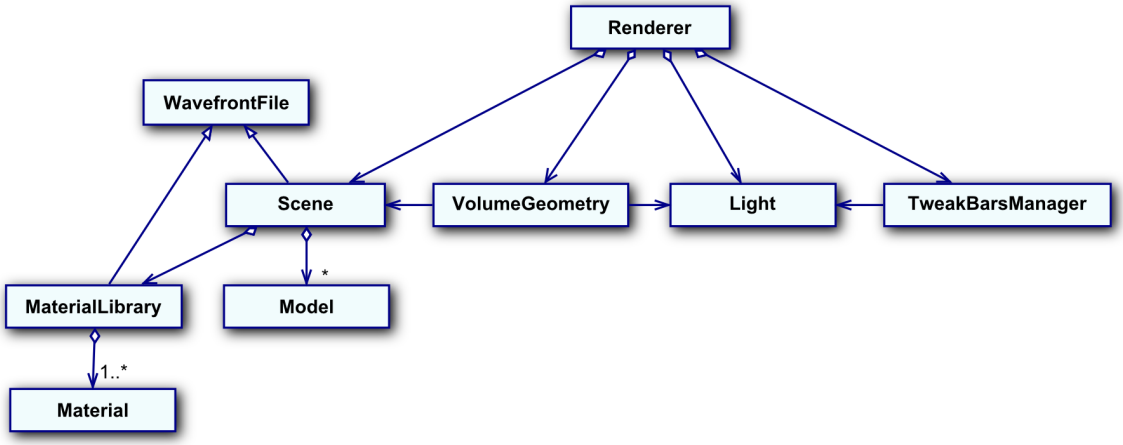
Rys. 34. Diagram klas aplikacji demonstracyjnej.

Klasa `MainConfig` przechowuje konfigurację programu wczytaną z pliku *Shadows.cfg*. Klasa ta dziedziczy po klasie `Config` pochodzącej z zewnętrznej biblioteki libconfig (zob. [17]). Takie podejście zostało zastosowane w celu uproszczenia oraz dostosowania do wymagań aplikacji dostępnego już interfejsu biblioteki. Najważniejszymi dla działa programu są klasy okien SetupWindow oraz GLWindow. Obie dziedziczą po abstrakcyjnej klasie `GLWindowEssentials`, która implementuje mechanizm obiektowego „opakowywania” (ang. *wrapping*) interfejsu okien dostępnego poprzez WinAPI (zob. [18]). Jest to konieczne ponieważ podobnie jak każda biblioteka systemowa systemu Windows, także i ta posiada interfejs napisany w stylu języka C. Wiąże to się z faktem, że funkcje zwrotne (ang. *callback*) mają sygnaturę niezgodną z sygnaturą zwykłych metod klas. Funkcje WinAPI wymagają podania wskaźnika do procedury obsługi komunikatów okna i jeżeli klasa okna ma implementować metodę pełniącą rolę tej procedury konieczne jest aby była to metoda statyczna. Metoda statyczna jest wspólna dla wszystkich egzemplarzy danej klasy i tym samym nie ma dostępu do wskaźnika `this`. Z tego powodu metoda statyczna `WndProcRouter` została tak zaprojektowana, aby przekierowywała przychodzące z systemu komunikaty do odpowiedniego obiektu okna. W związku z tym każda klasa dziedzicząca po abstrakcyjnej klasie `GLWindowsEssentials` musi implementować procedurę okna

WndProc. Klasa GLWindowsEssentials posiada również kilka podstawowych metod, które są pomocne podczas korzystania z biblioteki OpenGL. Aplikacja posiada dwa okna SetupWindow oraz GLWindow. Pierwsze z nich SetupWindow pozwala na konfigurację trybu wyświetlania głównego okna zawierającego trójwymiarową scenę generowaną z użyciem OpenGL. Zanim wybór odpowiednich opcji zostanie mu zaprezentowany sprawdzona jest dostępność poszczególnych ustawień w oparciu o możliwości danego komputera. W następnej kolejności z klasy MainConfig pobierana jest poprzednia konfiguracja wczytana z pliku, która o ile jest dostępna i zgodna z możliwościami systemu staje się konfiguracją domyślną. Zgodnie z tym co zostało wcześniej wspomniane dostęp do klasy MainConfig realizowany jest poprzez menedżera systemów SystemsManager. Klasa SetupWindow korzysta również z listy dostępnych trybów wyświetlania zapisanych w klasie DisplayModes. W drugim ze wspomnianych okien – GLWindow odbywa się właściwe rysowanie. Podczas jego tworzenia korzysta ono z ustawień wybranych przez użytkownika, a następnie zapisanych w klasie MainConfig. Główne okno aplikacji pełni rolę pośrednika pomiędzy systemem, a klasą Renderer odpowiedzialną za właściwe rysowanie oraz interakcję z użytkownikiem. Obiekt typu GLWindow przekazuje jej przychodzące z systemu komunikaty związane z rysowaniem, zdarzeniami klawiatury oraz myszy pozostawiając sobie obsługę pozostałych komunikatów i izolując tym samym je od reszty aplikacji.

W klasie Renderer (zob. rys. 35) zaimplementowane zostały wszystkie trzy omawiane techniki renderingu cieni. Podstawą jej działania jest metoda Draw, której działanie w ogólności ogranicza się do wywołania odpowiedniej funkcji rysującej wybranej w zależności od pożądanego sposobu rysowania sceny. Metoda DrawWithoutShadows tworzy obraz pozbawiony cieni, metoda DrawWithProjectedShadows implementuje rzutowanie cieni, metoda DrawWithStencilShadows realizuje algorytm cieni objętościowych, natomiast metoda DrawWithMappedShadows mapowanie cieni. Każda ze wspomnianych powyżej funkcji została wyposażona w szczegółowe komentarze opisujące implementację poszczególnych technik przy wykorzystaniu OpenGL, dlatego ich szczegółowe omówienie zostało w tekście pracy pominięte. Geometria obiektów sceny przechowywana jest w klasie Scene. Posiada ona metody umożliwiające wczytanie modeli z pliku w formacie Wavefront OBJ (zob. 5.3). Jeżeli obiekty mają powiązany ze sobą plik materiałów zostanie on również

wczytany za pośrednictwem klasy `MaterialLibrary`. Bardziej szczegółowy opis wspomnianych formatów plików został zawarty w punkcie 5.3. Obiekt `Scene` przechowuje listę wszystkich modeli oraz bibliotekę materiałów `MaterialLibrary`, która natomiast przechowuje listę wszystkich wykorzystywanych materiałów. Omawiany w niniejszej pracy problem generowania cieni nie miałby zastosowania jeżeli podczas rysowania nie był uwzględniany wpływ światła. Jego obsługa została zaimplementowana w klasie `Light`, która opakowuje oraz upraszcza interfejs dostępny z poziomu OpenGL. Po za sterowaniem parametrami samego światła klasa `Light` posiada metodę umożliwiającą narysowanie ikony prezentującej jego pozycję. Kolejnym elementem aplikacji potrzebnym do jej prawidłowego działania jest klasa `ShadowVolumes`. Metoda cieni objętościowych wymaga dość skomplikowanego przetwarzania obiektów sceny dlatego informacje o geometrii brył cieni zostały wydzielone w osobnej klasie. Metody tej klasy pozwalają na ich obliczenie zgodnie z zasadami przedstawionymi w punkcie 3.2. Korzysta ona przy tym z informacji pobranych z klas `Scene` oraz `Light`. Następnym bardzo istotnym elementem programu jest system okien pełniący rolę interfejsu użytkownika. Jest on obsługiwany przez klasę `TweakBarsManager` zbudowanej na bazie biblioteki AntTweakBars (zob. [19]). Otrzymywane z systemu komunikaty wejścia klawiatury oraz myszy są przekazywane z klasy `GLWindow` do klasy `Renderer`, która w pierwszej kolejności przekazuje je klasie `TweakBarsManager`. W przypadku ich nieobsłużenia klasa `Renderer` podejmuje się tego zadania, co ma np. miejsce w przypadku sterowania kamerą.



Rys. 35. Diagram klas części aplikacji odpowiedzialnej za rysowanie oraz interakcję z użytkownikiem.

Po za klasami widocznymi na diagramach z rys. 34 i 35 klasami aplikacja korzysta również z biblioteki log4cxx służącej do rejestrowania zdarzeń (zob. [20]). Wchodzi ona w

skład projektu Apache Logging Services związanego z najpopularniejszym, darmowym serwerem stron www Apache. Mimo swojego rodowodu doskonale nadaje się również do stosowania jej w innych typach aplikacji. W omawianym programie demonstracyjnym rejestrowanie wykorzystywane jest do zapisywania dziennika przechowującego informacje o błędach, które mogą wynikać m.in. ze źle utworzonego pliku sceny. Tym samym zaimplementowany został użyteczny mechanizm wykrywania nieprawidłowości mogących wystąpić podczas wczytywania danych. Dziennik znajduje się w pliku *shadows.log* tworzonym w głównym katalogu aplikacji.

Program do prawidłowego działania wymaga również biblioteki GLEW (zob. [21]), ułatwiającej sprawdzanie wersji biblioteki OpenGL oraz rozszerzeń dostępnych w danej jej implementacji. Użycie tej biblioteki upraszcza sposób pozyskiwania powyższych informacji o rozszerzeniach, które w przypadku korzystania ze standardowych funkcji WinAPI jest bardziej skomplikowane i mniej czytelne.

5.3. Format pliku Wavefront OBJ

Plik w formacie Wavefront OBJ zawiera informacje o zbiorze trójwymiarowych obiektów. Przechowuje informacje o ich wierzchołkach, powierzchniach oraz przypisanych im materiałach. Wszystkie te dane mają formę listy zapisanej w czytelnym także dla człowieka formacie tekstowym. Dzięki temu można go edytować przy pomocy zwykłego edytora tekstu, jak chociażby dostępnego w systemie Windows programu Notatnik. Format Wavefront OBJ jest bardzo rozbudowany (zob. [22]) i tylko część z jego możliwości jest wykorzystywana w mojej implementacji.

Jedną z głównych zalet formatu OBJ jest fakt, że jest on popularny, a w konsekwencji obsługiwany przez wiele programów służących do projektowania trójwymiarowych modeli, zarówno komercyjnych jak i darmowych. Dostępny za darmo program do modelowania i animacji o nazwie Blender (zob. [24]) posiada moduł eksportu geometrii do pliku OBJ. Przykład tak wygenerowanego pliku widoczny jest na poniższym listingu:

```
# Blender3D v247 OBJ File: pyramid.blend
# www.blender3d.org
mtllib pyramid.mtl
```

```
o Pyramid
v 0.707107 -1.000000 -0.707107
v 0.258819 -1.000000 0.965926
v -0.965926 -1.000000 -0.258819
v 0.000000 1.000000 -0.000000
v 0.000000 -1.000000 0.000000
vn 0.937086 0.242536 0.251091
vn -0.685995 0.242536 0.685994
vn -0.251091 0.242536 -0.937086
vn 0.000000 -1.000000 0.000000
usemtl PyramidMaterial
s 1
f 2//1 1//1 4//1
f 4//2 3//2 2//2
f 4//3 1//3 3//3
f 5//4 1//4 2//4
f 5//4 2//4 3//4
f 3//4 1//4 5//4
```

Powyższe linie opisują ostrosłup o podstawie trójkąta. Linie rozpoczynające się od znaku # to komentarze i podczas wczytywania pliku są pomijane. Nazwa pliku zawierającego definicję materiałów określona jest po identyfikatorze „`mtllib`”. W omawianym przykładzie jest to plik *pyramid.mtl*. Format tego pliku zostanie bardziej szczegółowo przedstawiony poniżej. Linia, w której pierwszym znakiem jest litera „`o`” rozpoczyna definicję danego obiektu; po literze „`o`” znajduje się logiczna nazwa obiektu. Współrzędne wierzchołka zapisywane są po identyfikatorze „`v`”, współrzędne normalnej – po „`vn`”, a koordynaty tekstur po – „`vt`”. Współrzędne tekstur celowo nie zostały wykorzystane w powyższym pliku ponieważ aplikacja pomimo, że potrafi je poprawnie odczytać to nie wykorzystuje ich podczas generowania obrazu. Możliwość załadowania ich z pliku OBJ pozwoli w przyszłości rozszerzyć program o nowe możliwości. Linie definiujące powierzchnie (odpowiadające prymitywom OpenGL) rozpoczynają się od znaku „`f`” i określają ich 3, 4 lub więcej wierzchołków. Mimo tego omawiana implementacja wspiera jedynie powierzchnie będące trójkątami, gdyż tylko takie mogą być wykorzystywane w zastosowanym w programie

algorytmie cieni objętościowych. Każdy z takich wierzchołków opisują 3 indeksy rozdzielone znakami „/”. Pierwszy to indeks wierzchołka (linii z pliku opisującej jego współrzędne), drugi jest indeksem współrzędnej tekstury, a ostatni indeksem normalnej. Możliwe są różne kombinacje występowania powyższych indeksów. Jeżeli wierzchołek określony jest jedynie przez swoje współrzędne wtedy podawany jest tylko jeden indeks (np. `f 2 1 4`), jeżeli posiada również współrzędną tekstury to występują dwa indeksy (np. `f 2/1 1/2 4/3`), natomiast gdy ma dodatkowo określoną normalną zapisane są trzy indeksy (np. `f 2/1/4 1/2/2 4/3/3`). Istnieje jeszcze jedna kombinacja tj. gdy znane są tylko współrzędne oraz normalna wierzchołka, co ma miejsce w zamieszczonym powyżej przykładowym pliku. W takim wariancie współrzędna tekstury jest pomijana, a pomiędzy dwoma skrajnymi indeksami pozostają dwa znaki „/” (np. `f 2//4 1//2 4//3`). Należy również zaznaczyć, że powyższe indeksy odnoszą się do globalnie indeksowanych w obrębie całego pliku list gdzie pierwszy wierzchołek, współrzędna tekstury czy normalna ma indeks równy 1. Dotychczas pominięte zostały jeszcze dwa typy obsługiwanych danych, którymi są nazwa materiału z pliku materiałów określonych w linii o identyfikatorze „`usemtl`” (w przykładzie nazwa materiału to `PyramidMaterial`) oraz numer grupy wygładzania – „`s`” (ang. *smoothing group*). W przypadku skryptu eksportującego w programie Blender numer grupy może być określony jedynie przez liczbę `1` co oznacza obiekt wygładzony (zob. rys. 2) lub w przypadku następującego po identyfikatorze tekstu „`off`” (np. „`s off`”) oznacza obiekt płaski.

Z plikiem OBJ zazwyczaj powiązany jest plik materiałów MTL. Przykładowy plik utworzony dla tej samej sceny co omawiany poprzednio plik materiałów został przdstawiony poniżej.

```
# Blender3D MTL File: pyramid.blend
# Material Count: 1
newmtl PyramidMaterial
Ns 96.078431
Ka 0.300000 0.000000 0.000000
Kd 0.000000 0.000000 0.640000
Ks 0.500000 0.500000 0.500000
Ni 1.000000
d 1.000000
```

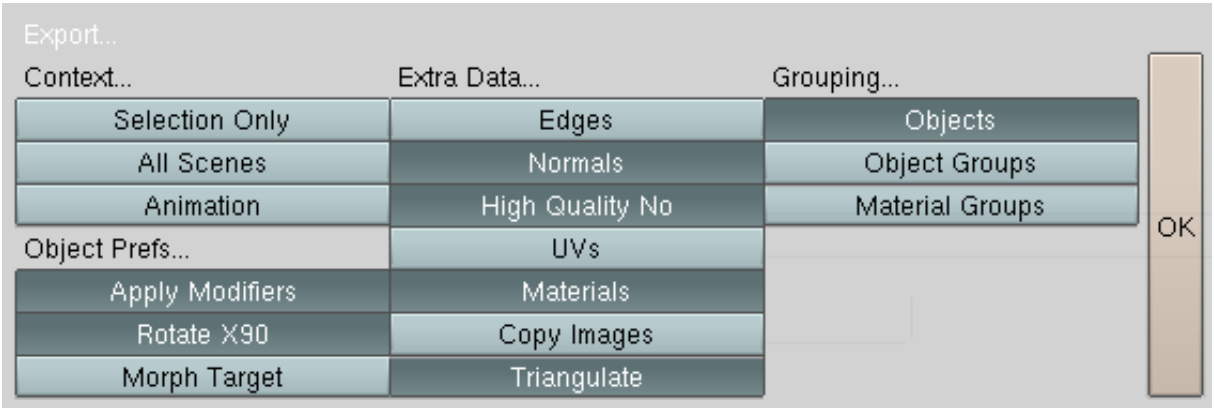
`illum 2`

Plik materiałów ma podobną strukturę jak plik OBJ. Definicja nowego materiału rozpoczyna się od podania jego nazwy następującej po identyfikatorze „`newmtl`”. Każdy materiał opisuje szereg parametrów, z których jedynie ograniczony podzbiór może mieć zastosowanie w OpenGL (w ramach podstawowych możliwości biblioteki). Materiał określa jego kolor dla światła otaczającego – identyfikator „`Ka`”, rozpraszanego – „`Kd`” oraz kolor refleksu – „`Ks`”. Parametr „`Ns`” jest wykładnikiem odbłysku będącym liczbą z zakresu 0 do 1000, która podczas wczytywania materiału jest konwertowana na wartość z zakresu 0 do 128 w celu zgodności ze specyfikacją OpenGL. Linia rozpoczynająca się literą „`d`” określa przezroczystość materiału i w niniejszej aplikacji określa wartość kanału alfa każdego z wymienionych powyżej kolorów. Pozostałe wyeksportowane z programu Blender informacje dotyczące materiałów nie są obsługiwane przez program ponieważ w większości nie mają zastosowania w przypadku modelu grafiki oferowanego przez bibliotekę OpenGL.

Aplikacja demonstracyjna zakłada jeszcze kilka specyficznych dla jej działania wymagań. Jednym z nich jest konieczność określenia płaszczyzny rzutowania dla metody rzutowania cieni, co zostało zrealizowane poprzez specjalny obiekt sceny o nazwie „`ShadowPlaneNS`”. Pierwszy z trójkątów wspomnianego obiektu posłuży do obliczenia równania płaszczyzny dla wspomnianej techniki tworzenia cieni i z tego względu wszystkie tworzące go powierzchnie (trójkąty) powinny leżeć w jednej płaszczyźnie. Sam obiekt „`ShadowPlaneNS`” nie jest rysowany dlatego powinien istnieć inny model pełniący rolę obiektu „przyjmującego” cień. Konieczny jest również podział na geometrię rzucająca i nie rzucającą cieni dlatego aplikacja interpretuje modele o nazwie kończącej się dużymi literami „`NS`” jako te, które nie rzucają cienia (od ang. *no shadow*). W przypadku dołączonej do programu przykładowej sceny takimi obiektem jest model „`WallsNS`” przedstawiający ściany pokoju. Gdyby ten obiekt rzucał cień to w przypadku światła kierunkowego wszystkie obiekty sceny byłyby spowite w mroku.

Podczas modelowania w programie Blender należy zwrócić uwagę na możliwość wystąpienia połączeń typu T, które polegają na tym, że krawędź jednego trójkąta sąsiaduje z dwoma innymi trójkątami. Niestety wspomniana aplikacja nie dysponuje odpowiednim narzędziem do usuwania takich niedoskonałości. W tej roli dobrze sprawdza się skrypt dostępny pod adresem [25] na forum BlenderArtists.com. Należy również pamiętać aby każdy

z obiektów tworzyła zamknięta geometria (czyli pozbawiona dziur). Ponadto, aby utworzona przy pomocy programu Blender scena była poprawnie rozpoznana przez aplikację demonstracyjną, należy zastosować parametry eksportu pliku OBJ przedstawione na rysunku 36.



Rys. 36. Ustawienia eksportu sceny do pliku OBJ w programie Blender.

6. Dodatki

Dodatek A. Współrzędne jednorodne

Punkty oraz wektory w bibliotece OpenGL jak i również w wielu innych pakietach do programowania grafiki trójwymiarowej reprezentowane są przy pomocy tzw. współrzędnych jednorodnych lub inaczej homogenicznych. Ich podstawową zaletą jest możliwość jednolitego traktowania wszystkich przekształceń wierzchołków i wyrażenia ich jako mnożenia przez odpowiednią macierz transformacji (zob. dodatek B). Ponadto za ich pomocą można przedstawić punkty w nieskończoności czyli wektory kierunkowe.

We współrzędnych jednorodnych punkt w n–wymiarowej przestrzeni przedstawiony jest za pomocą $n+1$ współrzędnych. Dodatkowa liczba w pełni rolę współczynnika skalowania pozostałych współrzędnych. Punkt w przestrzeni trójwymiarowej można przedstawić przy pomocy współrzędnych jednorodnych w następujący sposób:

$$A=\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}. \tag{60}$$

W przypadku gdy $w\neq 0$ odpowiada on punktowi w przestrzeni euklidesowej:

$$A=\begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix}. \tag{61}$$

W przeciwnym zaś razie tj. gdy jest $w=0$ to punkt w nieskończoności w kierunku x, y, z:

$$A=\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} \tag{62}$$

Trójwymiarowy punkt we współrzędnych jednorodnych reprezentuje prosta przechodząca przez środek czterowymiarowego układu współrzędnych. Współrzędne x, y, z rzutu środkowego tej prostej na płaszczyznę $w=1$ odpowiadają współrzędnym x, y, z tego punktu w przestrzeni euklidesowej. Oznacza to, że jeden punkt w przestrzeni trójwymiarowej można przedstawić za pomocą nieskończenie wielu czwórek współrzędnych:

$$A=\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}=\begin{bmatrix} \alpha x \\ \alpha y \\ \alpha z \\ \alpha w \end{bmatrix} \tag{63}$$

Mając na uwadze powyższe stwierdzenia, podstawowe działania na wektorach we współrzędnych jednorodnych wykonuje się nieco odmiennie niż we współrzędnych euklidesowych. Dodawanie dwóch wektorów wymaga w pierwszej kolejności zrzutowania ich na płaszczyznę $w=1$, a dopiero po tym można wykonać dodawanie:

$$\vec{a}+\vec{b}=\begin{bmatrix} a_x \\ a_y \\ a_z \\ a_w \end{bmatrix}+\begin{bmatrix} b_x \\ b_y \\ b_z \\ b_w \end{bmatrix}=\begin{bmatrix} a_x/a_w \\ a_y/a_w \\ a_z/a_w \\ 1 \end{bmatrix}+\begin{bmatrix} b_x/b_w \\ b_y/b_w \\ b_z/b_w \\ 1 \end{bmatrix}=\begin{bmatrix} a_x b_w+b_x a_w \\ a_y b_w+b_y a_w \\ a_z b_w+b_z a_w \\ a_w b_w \end{bmatrix}. \tag{64}$$

Skalowanie realizuje się następująco:

$$\alpha\cdot\vec{a}=\alpha\begin{bmatrix} a_x \\ a_y \\ a_z \\ a_w \end{bmatrix}=\begin{bmatrix} \alpha a_x \\ \alpha a_y \\ \alpha a_z \\ a_w \end{bmatrix}=\begin{bmatrix} a_x \\ a_y \\ a_z \\ a_w/\alpha \end{bmatrix}. \tag{65}$$

Mnożenie skalarne dwóch wektorów z przestrzeni euklidesowej zapisanych przy pomocy współrzędnych jednorodnych wykonuje się przy zachowaniu tych samych zasad co dodawanie:

$$\vec{a}\circ\vec{b}=\begin{bmatrix} a_x \\ a_y \\ a_z \\ a_w \end{bmatrix}\circ\begin{bmatrix} b_x \\ b_y \\ b_z \\ b_w \end{bmatrix}=\frac{a_x}{a_w}\cdot\frac{b_x}{b_w}+\frac{a_y}{a_w}\cdot\frac{b_y}{b_w}+\frac{a_z}{a_w}\cdot\frac{b_z}{b_w}=\frac{a_x b_x+a_y b_y+a_z b_z}{a_w b_w}. \tag{66}$$

Dodatek B. Przekształcenia macierzowe

Do opisu przekształceń obiektów w przestrzeni trójwymiarowej powszechnie stosuje się macierze. Za ich pomocą można opisać transformacje punktów takie jak obroty, skalowanie, pochylenia. Dzięki współrzędnym jednorodnym można za pomocą macierzy zapisać także przesunięcia, a nawet rzutowanie. To umożliwia traktowanie wszystkich podstawowych operacji na wierzchołkach jako mnożenia wektora położenia przez odpowiednią macierz:

$$M \cdot \vec{a} = \vec{a}\,' . \tag{67}$$

W przypadku przestrzeni trójwymiarowej macierz transformacji we współrzędnych jednorodnych ma wymiary 4x4. Czteroelementowy wektor położenia zapisuje się kolumnowo. W rezultacie otrzymujemy następujące równanie na współrzędne punktu $\vec{a}\,'$:

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \cdot \begin{bmatrix} a_x \\ a_y \\ a_z \\ a_w \end{bmatrix} = \begin{bmatrix} m_{11} \cdot a_x + m_{12} \cdot a_y + m_{13} \cdot a_z + m_{14} \cdot a_w \\ m_{21} \cdot a_x + m_{22} \cdot a_y + m_{23} \cdot a_z + m_{24} \cdot a_w \\ m_{31} \cdot a_x + m_{32} \cdot a_y + m_{33} \cdot a_z + m_{34} \cdot a_w \\ m_{41} \cdot a_x + m_{42} \cdot a_y + m_{43} \cdot a_z + m_{44} \cdot a_w \end{bmatrix} . \tag{68}$$

Kolejną zaletą stosowania macierzy do opisu przekształceń obiektów jest możliwość łączenia serii przekształceń w jedno. Jest to równoważne mnożeniu macierzy przekształceń składowych. W wyniku uzyskujemy pojedynczą macierz, co upraszcza obliczenia w przypadku dokonywania takiej samej transformacji na wielu punktach. Jeżeli wektor $\vec{a}$ poddany zostanie transformacji M_1, następnie M_2 oraz kolejnym aż do M_n to wynikowy wektor $\vec{a}\,'$ można obliczyć w następująco:

$$M_n \cdot \ldots \cdot M_2 \cdot M_1 \cdot \vec{a} = \vec{a}\,' . \tag{69}$$

Z powyższej równości można wyznaczyć:

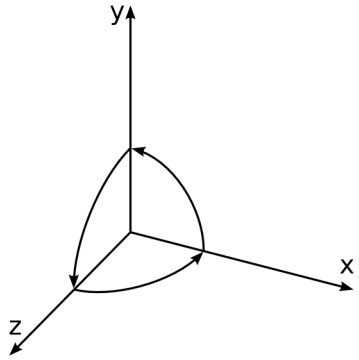
$$M = M_n \cdot \ldots \cdot M_2 \cdot M_1 , \tag{70}$$

i wtedy wzór (69) przyjmuje uproszczoną postać:

$$M \cdot \vec{a} = \vec{a}\,' . \tag{71}$$

Składanie przekształceń poza nielicznymi wyjątkami wymaga zachowania ich kolejności. Zastosowanie w pierwszej kolejności przesunięcia, a następnie obrotu da w rezultacie inny wynik niż translacja poprzedzona obrotem. Ponadto kształt macierzy transformacji zależy od wybranego układu współrzędnych i jest inny dla układu lewo- i prawoskrętnego. Biblioteka OpenGL wykorzystuje układ prawoskrętny (rys. 37) i taki jest stosowany w niniejszej pracy.

Układ lewoskrętny stosowany jest w bibliotece DirectX.



Rys. 37. Kartezjański, prawoskrętny układ współrzędnych z zaznaczonymi dodatnimi kierunkami obrotów wokół osi.

Jedną z podstawowych macierzy przekształcenia jest macierz skalująca współrzędne wierzchołka x, y oraz z. W każdym kierunku współczynnik skalowania może być inny np. s_x, s_y oraz s_z. Macierz tego przekształcenia ma następującą postać:

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{72}$$

Po wstawieniu tej macierzy do równania (67) można obliczyć wektor położenia nowego punktu:

$$S(s_x, s_y, s_z)\cdot\vec{a} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} a_x \\ a_y \\ a_z \\ a_w \end{bmatrix} = \begin{bmatrix} s_x\cdot a_x \\ s_y\cdot a_y \\ s_z\cdot a_z \\ a_w \end{bmatrix}. \tag{73}$$

W analogiczny sposób wyraża się również obroty. Macierze rotacji wokół głównych osi układu X, Y i Z oraz wyniki ich zastosowania przedstawiają odpowiednio równania (74-79):

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{74}$$

$$R_x(\theta)\cdot\vec{a} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} a_x \\ a_y \\ a_z \\ a_w \end{bmatrix} = \begin{bmatrix} a_x \\ \cos(\theta)\cdot a_y - \sin(\theta)\cdot a_z \\ \sin(\theta)\cdot a_y + \cos(\theta)\cdot a_z \\ a_w \end{bmatrix}, \tag{75}$$

$$R_y(\theta)=\begin{bmatrix}\cos(\theta) & 0 & \sin(\theta) & 0\\ 0 & 1 & 0 & 0\\ -\sin(\theta) & 0 & \cos(\theta) & 0\\ 0 & 0 & 0 & 1\end{bmatrix}, \tag{76}$$

$$R_y(\theta)\cdot\vec{a}=\begin{bmatrix}\cos(\theta) & 0 & \sin(\theta) & 0\\ 0 & 1 & 0 & 0\\ -\sin(\theta) & 0 & \cos(\theta) & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.\begin{bmatrix}a_x\\ a_y\\ a_z\\ a_w\end{bmatrix}=\begin{bmatrix}\cos(\theta)\cdot a_x+\sin(\theta)\cdot a_z\\ a_y\\ -\sin(\theta)\cdot a_x+\cos(\theta)\cdot a_z\\ a_w\end{bmatrix}, \tag{77}$$

$$R_z(\theta)=\begin{bmatrix}\cos(\theta) & -\sin(\theta) & 0 & 0\\ \sin(\theta) & \cos(\theta) & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}, \tag{78}$$

$$R_z(\theta)\cdot\vec{a}=\begin{bmatrix}\cos(\theta) & -\sin(\theta) & 0 & 0\\ \sin(\theta) & \cos(\theta) & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.\begin{bmatrix}a_x\\ a_y\\ a_z\\ a_w\end{bmatrix}=\begin{bmatrix}\cos(\theta)\cdot a_x-\sin(\theta)\cdot a_y\\ \sin(\theta)\cdot a_x+\cos(\theta)\cdot a_y\\ a_z\\ a_w\end{bmatrix}. \tag{79}$$

Wszystkie z dotychczas przedstawionych przekształceń możnaby zapisać bez wykorzystania współrzędnych jednorodnych. Siła takiego rozwiązania ujawnia się dopiero przy opisie przesunięć. We współrzędnych jednorodnych macierz translacji wyraża się następująco:

$$T(d_x,d_y,d_z)=\begin{bmatrix}1 & 0 & 0 & d_x\\ 0 & 1 & 0 & d_y\\ 0 & 0 & 1 & d_z\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{80}$$

co po podstawieniu do równania transformacji daje w rezultacie:

$$T(d_x,d_y,d_z)\cdot\vec{a}=\begin{bmatrix}1 & 0 & 0 & d_x\\ 0 & 1 & 0 & d_y\\ 0 & 0 & 1 & d_z\\ 0 & 0 & 0 & 1\end{bmatrix}.\begin{bmatrix}a_x\\ a_y\\ a_z\\ a_w\end{bmatrix}=\begin{bmatrix}a_x+d_x\cdot a_w\\ a_y+d_y\cdot a_w\\ a_z+d_z\cdot a_w\\ a_w\end{bmatrix}. \tag{81}$$

Gdy punkt jest znormalizowany i współrzędna $a_w=1$, poprawność macierzy (80) widoczna jest na pierwszy rzut oka.

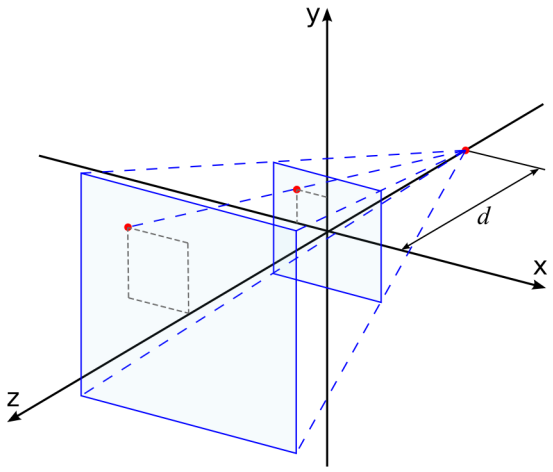
Kolejną kategorią przekształceń są projekcje. Rzut perspektywiczny na rzutnię znajdującą się w punkcie $z=0$ i środku rzutowania w punkcie $z=-d$ (rys. 38) opisuje następująca macierz:

$$M_{per}(d)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 1/d & 1\end{bmatrix} \tag{82}$$

rezultatem, której zastosowania jest nowy punkt $\vec{a}\,'$ obliczony jak poniżej:

$$M_{per}\cdot\vec{a}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 1/d & 1\end{bmatrix}\cdot\begin{bmatrix}a_x\\ a_y\\ a_z\\ a_w\end{bmatrix}=\begin{bmatrix}a_x\\ a_y\\ 0\\ 1/d\cdot a_z+a_w\end{bmatrix}. \tag{83}$$

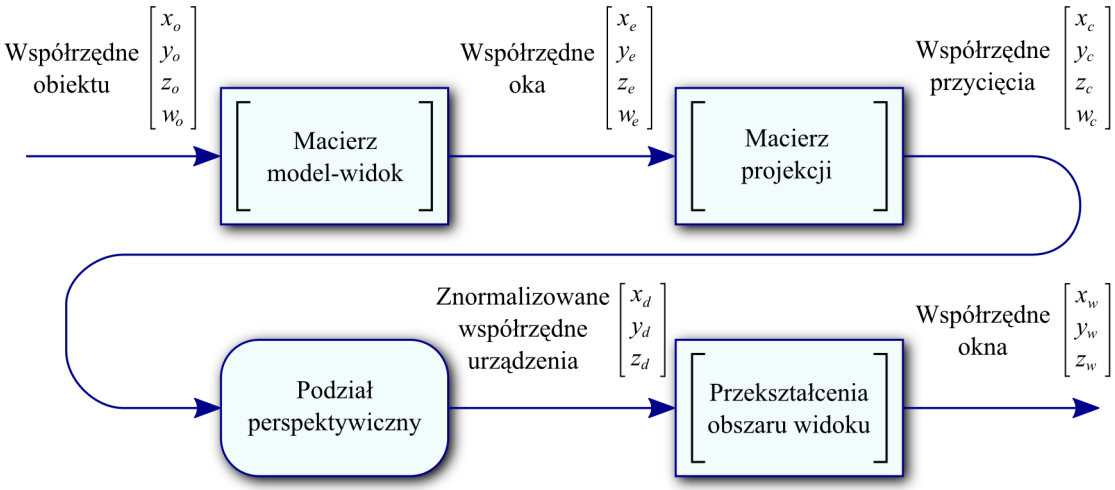
Można zauważyć, że w wyniku jej użycia skalowana jest współrzędna a_w w sposób proporcjonalny do wartości współrzędnej a_z wyrażającej odległość punktu od rzutni. Następstwem tego jest fakt, że im dalej rzutowany punkt znajduje się od rzutni tym mniejszą wartość mają współrzędne jego rzutu a'_x i a'_y.



Rys. 38. Rzut perspektywiczny punktu na rzutnię w punkcie $z=0$ i środku rzutowania w punkcie $z=-d$.

Dodatek C. Przekształcenia wierzchołków w bibliotece OpenGL

Przekształcenia wierzchołków w bibliotece OpenGL realizowane są w kilku etapach. Każdy z nich prowadzi do transformacji wektorów położenia punktów z jednego do drugiego układu współrzędnych. Proces ten można kontrolować za pomocą odpowiednich macierzy oraz funkcji API. Potok przekształceń wierzchołków OpenGL został przedstawiony na rys. 39.



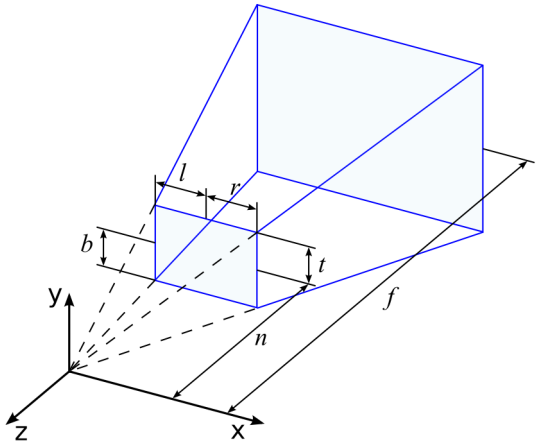
Rys. 39. Potok przekształceń wierzchołków w OpenGL

Wierzchołki modelu dostarczane są we współrzędnych obiektu (ang. *object coordinates*). Zazwyczaj są one wyrażone w układzie współrzędnych, którego początek znajduje się w środku obiektu. Aby ustawić model w odpowiednim miejscu sceny oraz poddać innym zamierzonym transformacjom takim jak obracanie, skalowanie lub np. pochylanie należy zastosować odpowiednią macierz model-widok. Ponadto macierz ta może realizować przekształcenia widoku (pozycji obserwatora) ponieważ z założenia obserwator znajduje się w środku układu współrzędnych. Jest to możliwe ze względu na fakt, że przesuwanie sceny i obserwatora są tak naprawdę tymi samymi transformacjami lecz wykonywane w odwrotnym kierunku. W wyniku zastosowania macierzy model-widok wierzchołki reprezentowane są we współrzędnych oka (ang. *eye coordinates*), których punkt $(0,0,0)$ wyznacza pozycję obserwatora. Następnie dokonywane jest rzutowanie wykorzystujące macierz projekcji. Jeżeli ma uwzględniać zjawisko perspektywy, można ją ustawić przy pomocy funkcji OpenGL `glFrustum(GLdouble l, GLdouble r, GLdouble b, GLdouble t, GLdouble n, GLdouble f)`. W takim przypadku

macierz ta przyjmuje postać (84).

$$M_{proj}(l,r,b,t,n,f)=\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{84}$$

Określa ją kilka zmiennych czyli odległość od obserwatora bliższej płaszczyzny obcinającej n (ang. *near plane*), dalszej f (ang. *far plane*) oraz pozycje krawędzi lewej l, prawej r, dolnej b i górnej t prostokąta leżącego w płaszczyźnie renderowanego obrazu. Powyższa macierz projekcji dokonuje przekształcenia perspektywicznego (ale bez rzutu) podobnie jak macierz (83) jednak pozwala dokładniejsze sprecyzowanie jego parametrów (rys. 40).



Rys. 40. Frusta ograniczająca obszar rysowania.

W rezultacie zastosowania omawianego przekształcenia (85) otrzymuje się punkt $\vec{V}_c$ we współrzędnych przycięcia (ang. *clip coordinates*).

$$\vec{V}_c = M_{proj}\cdot\vec{V}_e = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix} = \begin{bmatrix} \frac{2nx_e+(r+l)y_e}{r-l} \\ \frac{2ny+(t+b)z}{t-b} \\ -\frac{(f+n)z_e+2fnw_e}{f-n} \\ -z_e \end{bmatrix} \tag{85}$$

Jak można zauważyć otrzymany w wyniku tej operacji wierzchołek nie leży w płaszczyźnie przestrzeni euklidesowej $w=1$ (oprócz przypadku gdy $z=-1$). OpenGL wykorzystuje ten

fakt do eliminacji punktów znajdujących się po za przestrzenią przycięcia wyznaczaną przez obszar frusty (rys. 40). We współrzędnych przycięcia ostrosłup ten staje się prostopadłościanem, co znacznie upraszcza sprawdzanie, czy testowany punkt jest wewnątrz, czy poza frustą. Do tego celu służą poniższe nierówności:

$$\begin{aligned} -w_c \le x_c \le w_c \\ -w_c \le y_c \le w_c \\ -w_c \le z_c \le w_c \end{aligned} . \tag{86}$$

W następnym etapie potoku przekształceń nazywanym podziałem perspektywicznym biblioteka dokonuje dzielenia współrzędnych x_c, y_c, z_c wierzchołka $\vec{V}_c$ przez wartość współrzędnej w_c tak aby został on wyrażony w postaci (87).

$$\vec{V}_d = \begin{bmatrix} x_c / w_c \\ y_c / w_c \\ z_c / w_c \end{bmatrix} = \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix} \tag{87}$$

Tym samym liczba współrzędnych zostaje zredukowana do trzech co odpowiada już przestrzeni euklidesowej. W przypadku OpenGL wartości te zawierają się w zakresie od -1 do 1. Od tej chwili ma się do czynienia ze znormalizowanymi współrzędnymi urządzenia (ang. *normalized device coordinates*), które w kolejnym kroku przekształcenia obszaru widoku są odwzorowywane na współrzędne ekranu (ang. *window coordinates*) w pikselach. Transformacja ta sparametryzowana jest poprzez szerokość, wysokość oraz położenie lewego, dolnego rogu okna widoku (ang. *viewport*). Innymi słowy następuje w tym momencie dopasowanie (rozciągnięcie) obszaru przedniej ściany frusty do rozmiaru okna lub ekranu.

Dodatek D. Bufor szablonowy

Bufor szablonowy (ang. *stencil buffer*) jest pomocniczym buforem o rozmiarze takim samym jak bufor kolorów (tzn. jak wielkość obszaru okna przeznaczonego do wyświetlania obrazu generowanego przez OpenGL) oraz w zależności od implementacji o głębokości od 1 do 8 bitów. Jego zawartość nie jest widoczna na ekranie lecz może posłużyć do wykonywania operacji logicznych, których rezultat określa czy dany fragment przejdzie lub nie test szablonowy.

Podobnie jak bufor kolorów czy głębokości przed użyciem bufora szablonowego należy go wyczyścić przy pomocy instrukcji `glClear(GL_STENCIL_BUFFER_BIT)`. Domyślnie działanie tej procedury powoduje wypełnienie bufora zerami. Standardową wartość można zmienić podając własną liczbę całkowitą jako parametr funkcji `glClearStencil(GLint s)`. Włączenie testu szablonowego wykonuje instrukcja `glEnable(GL_STENCIL_TEST)`, a wyłączenie analogicznie `glDisable(GL_STENCIL_TEST)`. Do kontrolowania jego zachowania służy funkcja `glStencilFunc(GLenum func, Glint ref, Gluint mask)`. Jej parametry ustawiają funkcję porównującą (`func`) na jedną ze stałych pokazanych w tabeli 2, wartość referencyjną (`ref`) oraz maskę binarną (`mask`) nakładaną operacją logiczną `AND` na wartość znajdującą się w szablonie i wartość referencyjną.

Tab. 2. Funkcje porównujące dla testu szablonu

Wartość	**Funkcja porównująca testu szablonu**
`GL_NEVER`	Wynik testu zawsze będzie negatywny
`GL_ALWAYS`	Wynik testu zawsze będzie pozytywny
`GL_LESS`	Wartość referencyjna < wartość szablonu
`GL_LEQUAL`	Wartość referencyjna <= wartość szablonu
`GL_EQUAL`	Wartość referencyjna == wartość szablonu
`GL_GEQUAL`	Wartość referencyjna >= wartość szablonu
`GL_GREATER`	Wartość referencyjna > wartość szablonu
`GL_NOTEQUAL`	Wartość referencyjna != wartość szablonu

Sposób wprowadzania zmian do bufora definiuje się przy pomocy funkcji `void glStencilOp(GLenum fail, Glenum zfail, Glenum zpass)`. Jej

parametrami są wartości z tabeli 3. Określają one w jaki sposób mają się zmieniać wartości bufora szablonowego gdy:

- test szablonowy się nie powiedzie (`fail`),
- test szablonowy się powiedzie, a test głębokości nie (`zfail`)
- test szablonowy i test głębokości dadzą wynik pozytywny (`zpass`).

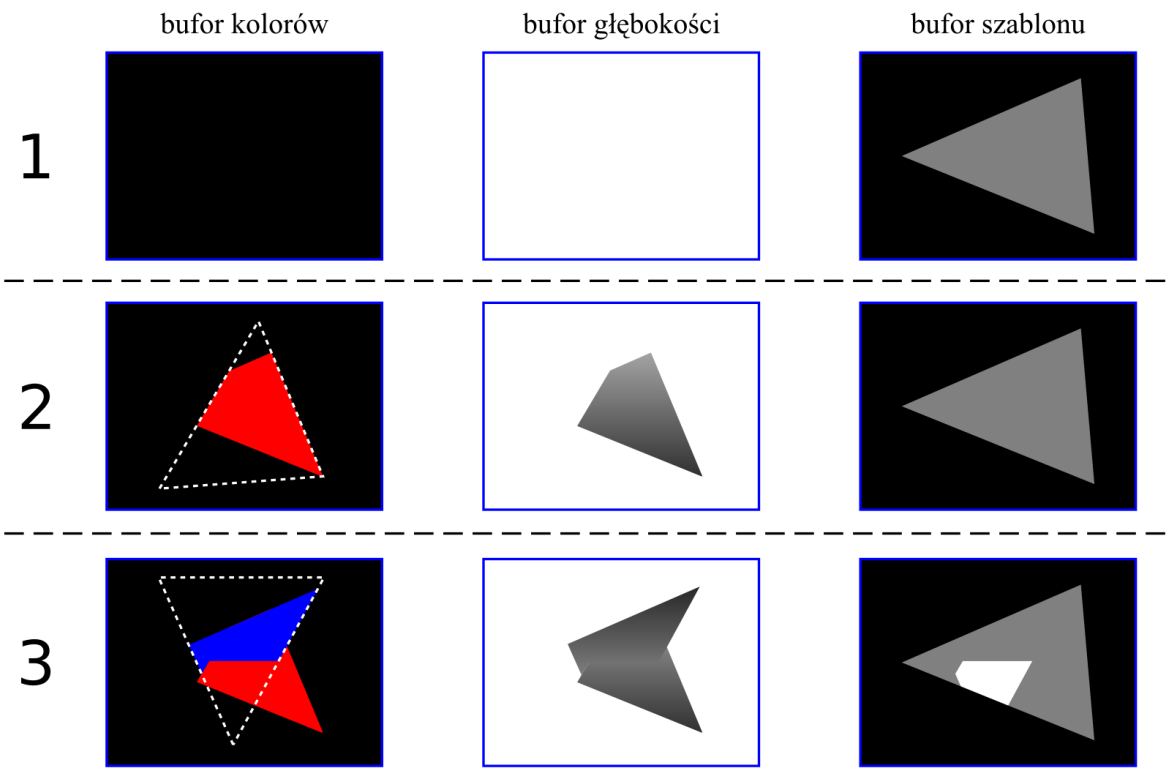
Tab. 3. Możliwe operacje dokonywane na wartościach w buforze szablonu

Wartość	**Operacja na szablonie**
`GL_KEEP`	Zachowuje aktualną wartość
`GL_ZERO`	Ustawia wartość na zero
`GL_REPLACE`	Zastępuje aktualną wartość wartością referencyjną określoną w funkcji `glStencilFunc()`
`GL_INCR`	Inkrementuje aktualną wartość
`GL_DECR`	Dekrementuje aktualną wartość
`GL_INVERT`	Odwraca bitowo aktualną wartość
`GL_INCR_WRAP`	Inkrementuje aktualną wartość i w razie przekroczenia maksymalnej wartości przyjmuje wartość zero (tylko w OpenGL 1.4 i nowszym)
`GL_DECR_WRAP`	Dekrementuje aktualną wartość i w razie przekroczenia wartości zero przyjmuje wartość maksymalną (tylko w OpenGL 1.4 i nowszym)

Przykład zastosowania tego narzędzia przedstawia rys. 41. Pokazane zostały na nim 3 kolejne stany buforów rysowania dla tego samego ustawienia, które zakłada, że:

- test szablonu przechodzą tylko te fragmenty, którym w buforze szablonu odpowiada wartość 1,
- powodzenie testu szablonu oraz jednoczesne niepowodzenie testu głębokości zwiększa wartość w szablonie, a pozostałe przypadki na mają na nią wpływu.

Pierwszy wiersz na rysunku przedstawia początkowy stanu renderingu. Drugi prezentuje rysowanie trójkąta, którego podstawa skierowana jest w stronę obserwatora. Na wynikowym obrazie pojawia się jedynie ta część figury, której odpowiada wartość 1 w buforze szablonu. W kolejnym etapie renderowania tworzony trójkąt zostaje dodatkowo ograniczony przez test głębi i ta jego część będąca za wcześniej utworzonym obiektem nie pojawia się na wynikowym obrazie. Ponadto dla tego obszaru gdzie dodatkowo wartość szablonu jest równa 1 dokonywane jest jej zwiększenie.



Rys. 41. Działania bufora szablonowego dla ustawień `glStencilFunc(GL_EQUAL, 1, 1)` oraz `glStencilOp(GL_KEEP, GL_INCR, GL_KEEP)` – linią przerywaną zaznaczony jest kontur aktualnie rysowanego trójkąta, , w z-buforze jaśniejszy odcień szarości odpowiada dalszemu punktowi, wartości w buforze szablonu zostały przedstawione za pomocą kolorów (0 – czarny, 1 – szary, 2 - biały)

7. Bibliografia

[1] *Nowa encyklopedia powszechna PWN*, PWN, Warszawa 1995

[2] Richard S. Wright Jr., Benjamin Lipchak, *OpenGL - księga eksperta*, Helion, Gliwice 2005.

[3] Marceli Stark, *Geometria analityczna*, http://matwbn.icm.edu.pl/kstresc.php?tom=26&wyd=10

[4] http://www.opengl.org/registry/doc/glspec21.20061201.pdf – OpenGL 2.1 Specification

[5] Jason Bestimt, *Real-Time Shadow Casting Using Shadow Volumes*, http://www.gamasutra.com/features/19991115/bestimt_freitag_01.htm

[6] http://www.opengl.org/registry/specs/NV/depth_clamp.txt – GL_NV_depth_clamp specification

[7] http://www.opengl.org/registry/specs/EXT/stencil_two_side.txt – GL_EXT_stencil_two_side specification

[8] http://www.opengl.org/registry/specs/EXT/stencil_wrap.txt – GL_EXT_stencil_wrap specification

[9] Morgan McGuire, GPU Gems – *Chapter 9. Efficient Shadow Volume Rendering,* http://http.developer.nvidia.com/GPUGems/gpugems_ch09.html

[10] Eric Lengyel, *Mathematics for 3D Game Programming and Computer Graphics*, Charles River Media, Inc., Hingham Massachusetts 2004.

[11] *ShaderX2: Introductions & Tutorials with DirectX 9*, Red. Wolfgang F. Engel, Wordware Publishing, Inc., Plano, Texas 2004.

[12] *ShaderX2: Shader Programming Tips & Tricks with DirectX 9*, Red. Wolfgang F. Engel, Wordware Publishing, Inc., Plano, Texas 2004.

[13] http://opengl.org/registry/specs/ARB/texture_non_power_of_two.txt – GL_ARB_texture_non_power_of_two specification

[14] http://www.opengl.org/registry/specs/EXT/frame_buffer_object.txt – GL_ARB_frame_buffer_object specification

[15] http://ati.amd.com/developer/gdc/2006/Isidoro-ShadowMapping.pdf – Isidoro, J. R. Shadow Mapping: GPU-based Tips and Techniques. Conference Session. GDC 2006. March 2006, San Jose, CA.

[16] Scott Patterson, *Szkielet gry wykorzystujący składanie obiektów*, Perełki programowania

gier, Helion, Gliwice 2003, 57-67.

[17] http://www.hyperrealm.com/libconfig/ – libconfig – C/C++ Configuration File Library

[18] https://secure.codeproject.com/KB/mobile/ltweight.aspx – Bradley Manske, Window Wrapper for WinCE Win32 "Hello World" app

[19] http://www.antisphere.com/Wiki/tools:anttweakbar – AntTweakBar GUI library to tweak parameters of OpenGL and DirectX applications [AntWiki]

[20] http://logging.apache.org/log4cxx/index.html – log4cxx - Short introduction to Apache log4cxx

[21] http://glew.sourceforge.net/ – GLEW: The OpenGL Extension Wrangler Library

[22] http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/ – Object Files (.obj)

[23] http://local.wasp.uwa.edu.au/~pbourke/dataformats/mtl/ – MTL material format (Lightwave, OBJ)

[24] http://www.blender.org/ – strona domowa programu Blender

[25] http://blenderartists.org/forum/showthread.php?t=81516 – BlenderArtists.org forum, T Junction remover

[26] http://wikipedia.pl – Wolna encyklopedia.

[27] http://opengl.org – strona domowa biblioteki OpenGL.

[28] http://nehe.gamedev.net – NEHE Productions.