Rozdział 8
Wzory na macierze OpenGL

Jacek Matulewski, wersja 2014-03-07

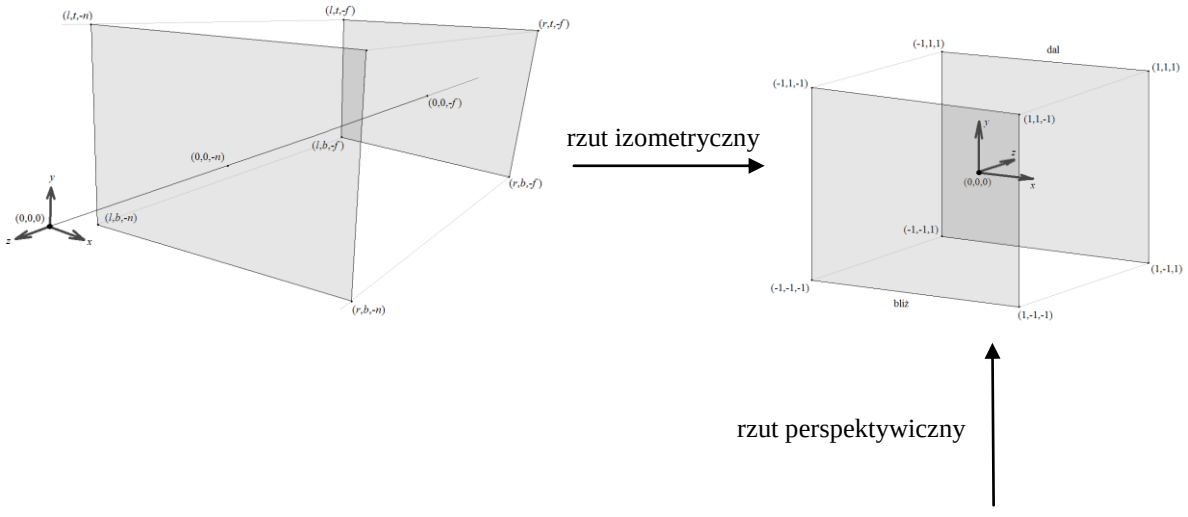
Dodać YawPitchRoll???

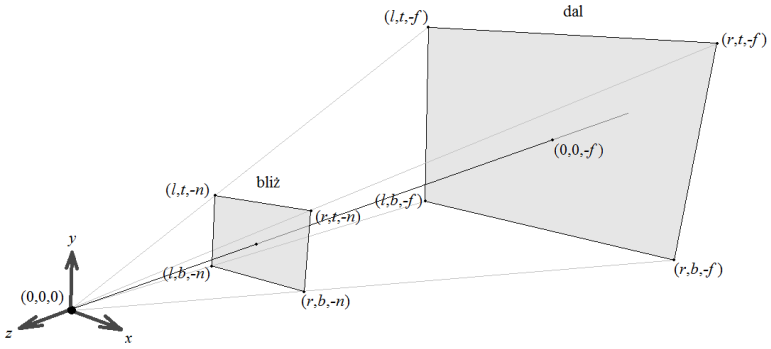
8->7

Dalszy rozwój aplikacji w trybie rdzennym nie jest już możliwy bez zaimplementowania operacji na macierzach. Możemy oczywiście użyć jednej z dostępnych w Internecie bibliotek, najlepiej GLM (zob. dodatek A), ale zgodnie z filozofią tej książki chciałbym, abyśmy odpowiednie rozwiązania przygotowali samodzielnie, a przy okazji zrozumieli stojącą za nimi matematykę.

Werteksy przesyłane do karty graficznej wyrażone powinny być w czterowymiarowym układzie współrzędnych modelu. Macierze świata i widoku, ewentualnie jedna macierz model-widok, transformują współrzędne werteksu z lokalnego układu obiektu do układu współrzędnych kamery – to jest ten układ, który zazwyczaj przedstawiany jest na rysunkach wprowadzających do OpenGL; z osią *OZ* skierowaną do widza, osią *OX* skierowaną w prawo i osią *OY* – do góry. Układ modelu może mieć natomiast dowolne położenie i orientację - macierze świata i widoku mogą być bowiem iloczynem macierzy opisujących obroty, translacje, skalowania, odbicia czy pochylenia. Te macierze omówione zostaną niżej w tym rozdziale. Za dalsze przekształcenia, tj. transformację z układu odniesienia kamery do układu przycinania odpowiada macierz rzutowania, od wyprowadzenia której zaczniemy. Wyprowadzenie nie jest matematycznie bardzo trudne, ale dość zawiłe. Warto zatem czytać następny fragment z ołówkiem w ręku.

Macierze rzutowania





Rysunek 8.1. Bryły widzenia w układzie kamery w przypadku perspektywy (*frustum*) i rzutu izometrycznego oraz po transformacji do układu NDC (sześcian w prawym górnym rogu)

Macierz rzutowania odpowiedzialna jest za transformację z układu współrzędnych kamery (lub jak kto woli – oka) do układu przycinania (ang. *clip coordinate system*). To nie jest jednak układ docelowy. Uzupełnieniem tego przekształcenia jest bowiem tzw. dzielenie perspektywiczne, tj. podzielenie współrzędnych *x*, *y* i *z* przez współrzędną skalowania *w*. W efekcie następuje transformacja do trójwymiarowego układu unormowanych współrzędnych urządzenia (NDC), w którym bryła widzenia przyjmuje kształt sześcianu (rysunek 8.1). Kształt sceny wyznacza charakter rzutowania, a dokładniej macierz odwrotna do macierzy rzutowania oraz to, że w układzie NDC ma ona mieć kształt sześcianu. A zatem w przypadku rzutowania izometrycznego, w którym przekształcenia wszystkich współrzędnych są liniowe, bryła widzenia w układzie kamery będzie prostopadłościanem. W przypadku rzutu perspektywicznego – będzie miała kształt *frustum*.

Zgodnie z konwencją OpenGL przyjmijmy, że ekran ułożony jest w taki sposób, że w układzie odniesienia kamery znajduje się w płaszczyźnie prostopadłej do osi *OZ* (osi optycznej wirtualnej kamery), a jego lewa i prawa krawędź znajdują się w $x = l$ i $x = r$, natomiast dolna i górna w $y = b$ i $y = t$ (zwykle *l* i *b* są ujemne, a *r* i *t* – dodatnie, ale to nie jest konieczne). Ekran ustawiony jest o *n* od kamery, a tylna płaszczyzna ograniczająca scenę o *f*.[1] Ponieważ oś OZ skierowana jest w kierunku do widza z zerem w położeniu kamery (rysunek 8.1), ekran znajduje się w $z = -n$, a tylna płaszczyzna odcinania w $z = -f$. Wartości współrzędnej *z* przyjmują wartości ujemne (*n* i *f* są dodatnie). Te wartości, a więc kolejno *l*, *r*, *b*, *t*, *n* i *f* są argumentami funkcji `glFrustum` i `glOrtho` tradycyjnego OpenGL. Jak wspomniałem, w efekcie rzutowania i następującego po nim dzielenia przez współrzędną *w* uzyskujemy wartości współrzędnych w układzie NDC, w którym scena staje się sześcianem o boku 2. W tym układzie współrzędne *x*, *y* i *z* przyjmują wartości od -1 do 1, przy czym poszczególne współrzędne są transformowane w taki sposób, aby zachodziły relacje:

(8.1)

$$x: [l, r] \rightarrow [-1,1]$$
$$y: [b, t] \rightarrow [-1,1]$$
$$z: [-n, -f] \rightarrow [-1,1]$$

Zwróćmy uwagę, że w przypadku współrzędnej *z* zmianie ulega kierunek osi *OZ* (pamiętajmy, że $f > n$). W układach przycinania i NDC wartość współrzędnej *z* oddaje bowiem głębię, czyli odległość werteksu od kamery, i rośnie, gdy przesuwamy się od ekranu do tylnej „ściany” frustum. Przekształcenia wszystkich współrzędnych z układu kamery do układu przycinania są liniowe, a więc zadane przez:

(8.2)

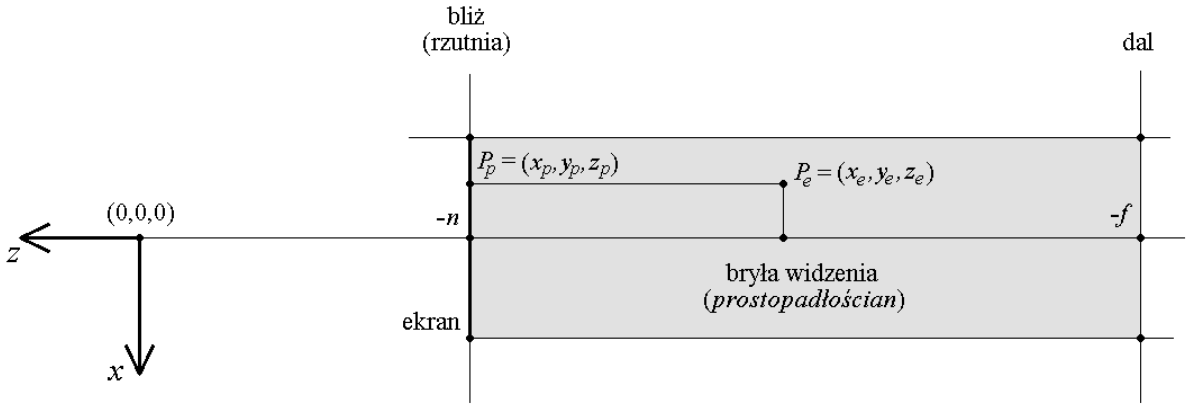
$$x_c = a_x x_e + b_x,$$
$$y_c = a_y y_e + b_y,$$
$$z_c = a_z z_e + b_z.$$

Uprzedzając fakty, zdradzę jednak, że o ile przekształcenie współrzędnych *x* i *y* do układu NDC także jest liniowe, to nie musi tak być w przypadku współrzędnej *z*, a mówiąc konkretniej: nie jest tak w przypadku rzutu perspektywicznego.

[1] Oznaczenia *l*, *r*, *b*, *t*, *n* i *f* biorą się od angielskich słów *left*, *right*, *bottom*, *top*, *near* i *far*, a więc kolejno lewy, prawy, dolny, górny, bliż i dal.

We wzorze (8.2) stosuję oznaczenia, w których indeks *c* oznacza współrzędną w układzie przycinania (ang. *clipping*), a *e* – w układzie kamery (od ang. *eye*, czyli oko). To typowa konwencja grafiki 3D, którą postanowiłem tutaj zachować, pomimo że od „oka" wolę używać terminu „kamera".

Macierz rzutowania izometrycznego



Rysunek 8.2. Rzut z kierunku +y na płaszczyznę OXZ

Wyprowadźmy najpierw wzór macierzy rzutowania izometrycznego. Po pierwsze dlatego, że jest prostsza – wszystkie przekształcenia są w niej liniowe, a po wtóre – wyprowadzenie jej można traktować jako wstęp do wyprowadzenia macierzy rzutowania perspektywicznego.

Skupmy się na początek na współrzędnej *x*. Wiemy, że dla wartości $x_e = l$ powinniśmy uzyskać $x_c = -1$, natomiast dla $x_e = r$ otrzymamy $x_c = 1$. Mamy zatem dwa fakty, które pozwolą nam znaleźć wartości stałych a_x i b_x:

(8.3)

$$-1 = a_x l + b_x,$$
$$1 = a_x r + b_x.$$

Z drugiego równania znajdujemy wartość stałej $b_x = 1 - a_x r$, którą wstawiamy do pierwszego, co pozwoli nam ustalić, że:

(8.4)

$$a_x = \frac{2}{r - l},$$
$$b_x = -\frac{r + l}{r - l}.$$

Jeżeli $r > l$, to stałą a_x jest dodatnia. Jest ona odpowiedzialna za przeskalowanie układów odniesienia – równa się wobec tego stosunkowi szerokości płaszczyzny odcinającej w bliży (ekranu) w układach przycinania (równa 2) i kamery (równa $r - l$). Druga stała opisuje różnicę położenia początku układu współrzędnych w obu układach – jeżeli r i l mają wartości symetryczne względem zera tj. $l = -r$, to b_x równa jest zeru. W efekcie transformacja przyjmuje postać:

(8.5)

$$x_c = \frac{2}{r - l} x_e - \frac{r + l}{r - l}.$$

Sprawdźmy, czy wszystko jest w porządku, podstawiając za x_e wartości l, $(l+r)/2$ i r. Uzyskamy kolejno: –1, 0 i 1. Analogiczne rozumowanie prowadzi do znalezienia wzoru na transformację współrzędnej *y*:

(8.6)

$$y_c = \frac{2}{t - b} y_e - \frac{t + b}{t - b}$$

i współrzędnej *z*:

(8.7)

$$z_c = -\frac{2}{f-n} z_e - \frac{f+n}{f-n}.$$

W tym trzecim przypadku minus przy pierwszym wyrazie związany jest ze zmianą kierunku osi OZ. W przypadku transformacji izometrycznej, dzielenie perspektywiczne nie powinno wprowadzać żadnych zmian. Dlatego współrzędna w_c powinna mieć wartość równą jedności. Z tego wynika, że transformacja do współrzędnych przycinania, a potem do współrzędnych NDC polega zatem jedynie na ich przeskalowaniu.

We wzorach (8.5)-(8.8) kryje się jednak pewien problem: jest nim wolny wyraz niezależący od żadnej współrzędnej układu kamery. Taki wyraz uniemożliwia zapis przekształcenia za pomocą macierzy! Sprawę ratuje jednak „nadmiarowa" współrzędna w. Jeżeli założymy, że w układzie kamery współrzędna w_e musi mieć wartość równą 1, to wówczas przekształcenie do współrzędnych przycinania i NDC (w przypadku rzutu izometrycznego te dwa układy są identyczne) można zapisać:

(8.8)

$$x_{NDC} = x_c = \frac{2}{r-l} x_e - \frac{r+l}{r-l} w_e,$$
$$y_{NDC} = y_c = \frac{2}{t-b} y_e - \frac{t+b}{t-b} w_e,$$
$$z_{NDC} = z_c = -\frac{2}{f-n} z_e - \frac{f+n}{f-n} w_e,$$

co z kolei można wyrazić w zwarty sposób za pomocą macierzy:

(8.9)

$$\hat{O} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Jeszcze raz przypomnę, że formalnie rzecz biorąc macierz ta przeprowadza współrzędne w układzie kamery do współrzędnych przycinania. Jednak w tym przypadku dzielenie perspektywiczne nie zmienia ich wartości, bo $w_c = w_e = 1$, zatem współrzędne NDC mają taką samą wartość.

Jeżeli założymy, że scena jest symetryczna względem osi optycznej kamery, co jest częste w praktyce, czyli $r = -l = w/2$ (wielkość $w = r - l$ to szerokość ekranu) oraz $t = -b = h/2$ ($h = t - b$ to wysokość ekranu), to macierz rzutowania izometrycznego znacznie się uprości:

(8.10)

$$\hat{O} = \begin{pmatrix} \frac{2}{w} & 0 & 0 & 0 \\ 0 & \frac{2}{h} & 0 & 0 \\ 0 & 0 & -\frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Zakładając dodatkowo, że $w = 2$ ($l = -1$, $r = 1$), $h = 2$ ($t = 1$, $b = -1$) i $n = 1$, uzyskamy postać:

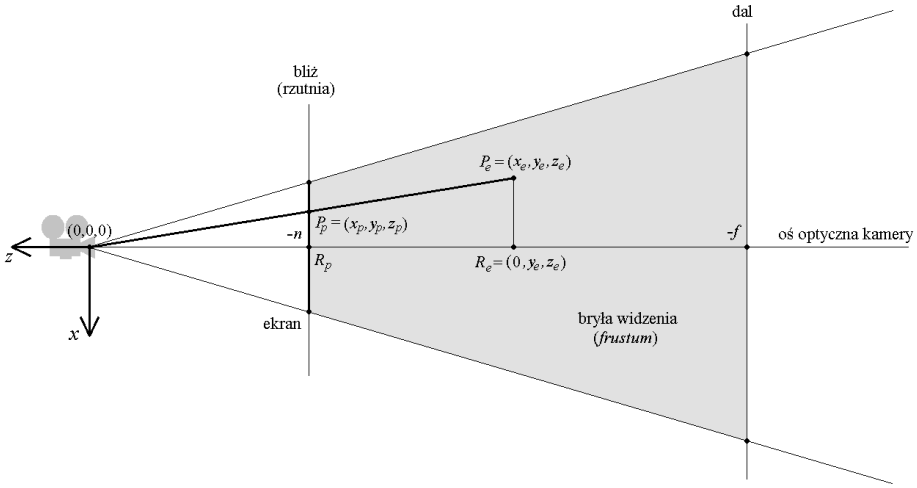
(8.11)

$$\hat{O} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{2}{f-1} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Ta postać jest bliższa macierzy (3.1), na której wyrabialiśmy sobie intuicję dotyczącą macierzy rzutowania. Największym różnicą jest pominięcie w macierzy (3.1) głębi (wyraz w trzecim wierszu) oraz odwrócony tu kierunek osi *OZ*.

Macierz rzutowania perspektywicznego

W rzutowaniu izometrycznym bryła widzenia ma kształt prostopadłościanu, zatem jej boczne krawędzie są do siebie równoległe i jednocześnie równoległe do osi *OZ*. W przypadku rzutu perspektywicznego jest inaczej – wszystkie te krawędzie leżą na prostych, które przecinają się w jednym punkcie nazywanym ogniskiem rzutowania. To punkt, w którym znajduje się kamera – początek układu współrzędnych kamery. W rzutowaniu izometrycznym położenie kamery jest umowne. Z istnienia ogniska i ze zbiegania promieni widzenia do jednego punktu wynika, że obiekt mniejszy, ale znajdujący się bliżej, może przesłaniać obiekt większy, ale dalszy (por. rysunek 8.3). Zastanówmy się wobec tego jak w miarę oddalania od kamery zmienia się wielkość obiektów, które na ekranie mają ten sam rozmiar albo odwrotnie: jak rozmiar obrazu na ekranie w rzucie perspektywicznym zależy od jego odległości od ekranu i kamery. W rzutowaniu izometrycznym rozmiar obrazu zrzutowanego na ekran obiektu, jest taki sam, jak samego werteksu – współrzędne dowolnego werteksu po zrzutowaniu, nazwijmy je x_p i y_p, mają te same wartości, co współrzędne x_e i y_e (rysunek 8.2).



Rysunek 8.3. Rzut frustum z kierunku +y. Zaznaczony werteks P_e nie musi znajdować się w płaszczyźnie OXZ zawierającej oś optyczną

Spójrzmy na kamerę i frustum „z góry” tj. z kierunku $+y$. Pokazuje to rysunek 8.3 prezentujący płaszczyznę *OXZ*, który pozwoli nam sprawdzić jak współrzędna x_p obrazu werteksu na ekranie powinna zależeć od współrzędnej z_e oryginalnego werteksu. Rozważmy w tym celu punkt P_e znajdujący się w obrębie sceny. Z podobieństwa trójkątów OP_eR_e i OP_pR_p wynika, że

(8.12)

$$\frac{x_e}{z_e} = \frac{x_p}{-n}.$$

Pamiętajmy, że z_e i $z_p = -n$ są mniejsze od zera. Te trójkąty nie muszą leżeć w płaszczyźnie *OXZ*, zatem długości odcinków OR_e i OR_p wcale nie są równe z_e i z_p, ale ich stosunek równy jest stosunkowi tych współrzędnych. Identyczne proporcje można zapisać dla zmiennej *y*. Z tego wynika, że zrzutowany na ekran obiekt ma rozmiary:

(8.13)

$$x_p = -n\frac{x_e}{z_e},$$

$$y_p = -n\frac{y_e}{z_e}.$$

Już teraz widać, że rozmiar ten maleje wraz ze zwiększaniem wartości $-z_e$, czyli odległości od kamery. To właśnie jest perspektywiczne pomniejszenie dalszych obiektów. Transformacja do układu NDC powinna je uwzględniać. Właśnie dlatego w przypadku rzutowania perspektywicznego przekształceniom (8.8) nie podlegają współrzędne x_e i y_e oryginalnego werteksu, a współrzędne x_p i y_p obrazu zrzutowanego na ekran.

Połączmy zatem oba przekształcenia. W tym celu musimy złożyć wzory (8.5)-(8.7). Uzyskamy w ten sposób transformację współrzędnych werteksu z układu kamery bezpośrednio do współrzędnych NDC:

(8.14)

$$x_{NDC} = \frac{2}{r-l}\left(-n\frac{x_e}{z_e}\right) - \frac{r+l}{r-l} = -\frac{1}{z_e}\left(\frac{2n}{r-l}x_e + \frac{r+l}{r-l}z_e\right),$$
$$y_{NDC} = \frac{2}{t-b}\left(-n\frac{y_e}{z_e}\right) - \frac{t+b}{t-b} = -\frac{1}{z_e}\left(\frac{2n}{t-b}x_e + \frac{t+b}{t-b}z_e\right),$$

Zwróćmy uwagę, że o ile we wzorach (8.14) zawartość nawiasów może być zapisana za pomocą macierzy z elementami, które zależą jedynie od parametrów frustum (stałe l, r, b i t), to współczynnik przed nawiasem zależy od współrzędnej transformowanego werteksu. Takiego przekształcenia nie zapiszemy za pomocą macierzy! A to oznacza, wbrew temu co się zwykle mówi, że sama macierz rzutowania perspektywicznego wcale nie umożliwia rzutowania perspektywicznego. Zasadniczą pracę w tym względzie musi wziąć na siebie dzielenie perspektywiczne, czyli redukcja współrzędnych jednorodnych w układzie przycinania do współrzędnych kartezjańskich NDC z jednoczesnym podzieleniem współrzędnych x_c, y_c i z_c przez współrzędną w_c. Aby to zadziałało we właściwy sposób musimy jednak zadbać o to, aby we współrzędnej w_c znalazła się wartość współrzędnej z_e. To oznacza, że wzory na przekształcenie układu współrzędnych do układu przycinania, a więc bez dzielenia perspektywicznego, powinny wyglądać następująco:

(8.15)

$$x_c = \frac{2n}{r-l}x_e + \frac{r+l}{r-l}z_e,$$
$$y_c = \frac{2n}{t-b}x_e + \frac{t+b}{t-b}z_e,$$
$$w_c = -z_e.$$

Jak widać do współrzędnej w_c przeniosłem także minus, który widoczny jest we wzorach (8.14). Nadal nie mamy jeszcze wzoru na transformację współrzędnej z. Wartość z_c nie zależy od x_e i y_e, jest liniową funkcją z_e (por. wzór 8.2):

(8.16)

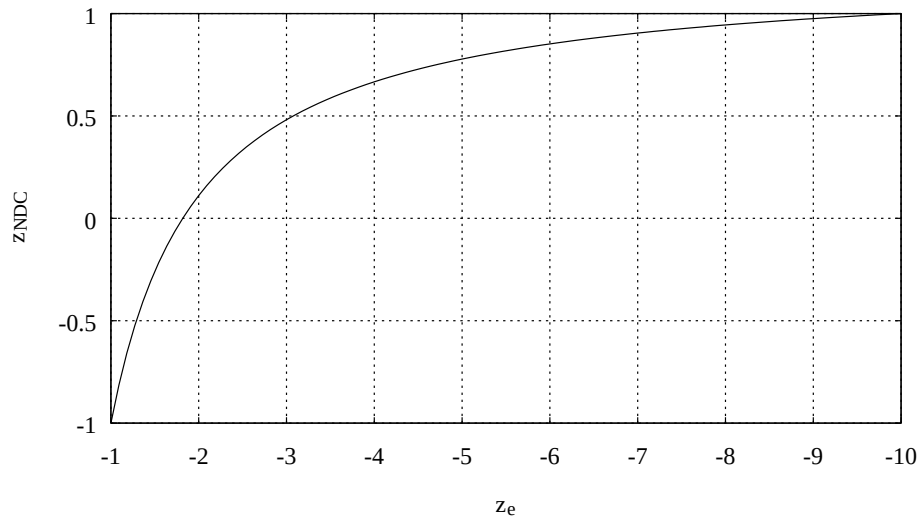
$$z_c = a_z z_e + b_z.$$

Musimy także pamiętać o tym, że przy przejściu do współrzędnych NDC wszystkie współrzędne, w tym także współrzędna z_c, zostanie podzielona przez $w_c = -z_e$, co jest konieczne, aby wprowadzić perspektywę:

(8.17)

$$z_{NDC} = -\frac{1}{z_e}(a_z z_e + b_z),$$

przy czym a_z musi być ujemne, bo oś OZ w układzie NDC zmienia kierunek. Przekształcenie współrzędnej z_e do z_{NDC}, w odróżnieniu od transformacji do układu przycinania (8.16), traci charakter liniowy (por. rysunek 8.4).



Rysunek 8.4. Współrzędna z_{NDC} w funkcji współrzędnej z_e z układu kamery do układu NDC dla $n = 1$ i $f = 10$

Ponadto wiemy, że dla $z_e = -n$ powinniśmy uzyskać $z_{NDC} = -1$, a dla $z_e = -f$ powinniśmy otrzymać $z_{NDC} = 1$. Podstawiając te wartości do wzoru (8.17) otrzymujemy:

(8.18)

$$-1 = -\frac{1}{(-n)}(a_z(-n) + b_z) = -a_z + \frac{b_z}{n},$$

$$1 = -\frac{1}{(-f)}(a_z(-f) + b_z) = -a_z + \frac{b_z}{f}.$$

Proste przekształcenia pozwalają na znalezienie wartości a_z i b_z:

(8.19)

$$a_z = -\frac{f+n}{f-n} < 0,$$

$$b_z = -\frac{2nf}{f-n}.$$

Transformacja współrzędnej z z układu kamery do układu współrzędnych NDC przyjmuje wobec tego postać

(8.20)

$$z_{NDC} = -\frac{1}{z_e}\left(-\frac{f+n}{f-n}z_e - \frac{2nf}{f-n}\right).$$

Nie usuwam minusów z tego wzoru, ponieważ minus przed nawiasem jest już zafiksowany w dzieleniu perspektywicznym (ustaliliśmy wcześniej, że $w_c = -z_e$). Pojawia się tu ponownie problem zwiazany z wolnym wyrazem wewnątrz nawiasu. I tym razem możemy problem rozwiązać zakładając, że współrzędna w_e w układzie kamery musi mieć wartość równą 1. Wówczas można napisać:

(8.21)

$$z_{NDC} = -\frac{1}{z_e}\left(-\frac{f+n}{f-n}z_e - \frac{2nf}{f-n}w_e\right).$$

W ten sposób ustaliliśmy przekształcenia wszystkich współrzędnych i możemy zapisać kompletny wzór na macierz rzutowania perspektywicznego:

(8.22)

$$\hat{P} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Taki właśnie wzór implementuje funkcja `glFrustum` tradycyjnego OpenGL, funkcja `frustum` z biblioteki GLM i taki wzór zaimplementujemy też w następnym rozdziale w naszej własnej klasie macierzy.

Sprawdźmy jeszcze jak przekształcenie (8.21) zmieni wartość współrzędnych $z_e = -n$, $-(n+f)/2$ i f z układu odniesienia kamery. Pamiętając o wykresie z rys. 8.3 nie należy spodziewać się, „ładnych" wyników dla wartości pośrednich między $-n$, a $-f$, w szczególności punkt o składowej z_e leżącej dokładnie pośrodku między płaszczyznami bliży i dali, w nowych współrzędnych nie będzie miał współrzędnej z_{NDC} równej zeru. Podstawiając te wartości z_e kolejno do wzoru (8.21) otrzymamy wyniki: –1, $(f-n)/(f+n)$ i +1. Wartość zerową z_{NDC} otrzymamy natomiast dla $z_e = -2nf/(f+n)$. Fakt, że przekształcenie współrzędnej z nie jest liniowe nie jest tak dużym problemem, jak mogłoby się wydawać. Po przeprowadzeniu transformacji, współrzędna ta służy bowiem jedynie do przeprowadzenia testu głębi. A do tego ważne jest zachowanie kolejności odległości punktów od kamery. Warunkiem jest więc jedynie, aby transformacja była zmienna monotonicznie, a nie aby zachowywała proporcje odległości punktów od kamery.

Podobnie, jak w przypadku macierzy rzutowania izometrycznego sprawdźmy jak wygląda macierz rzutowania perspektywicznego dla symetrycznie ustawionej sceny (stosuję te same oznaczenia, co w przypadku wzoru (8.10):

(8.23)

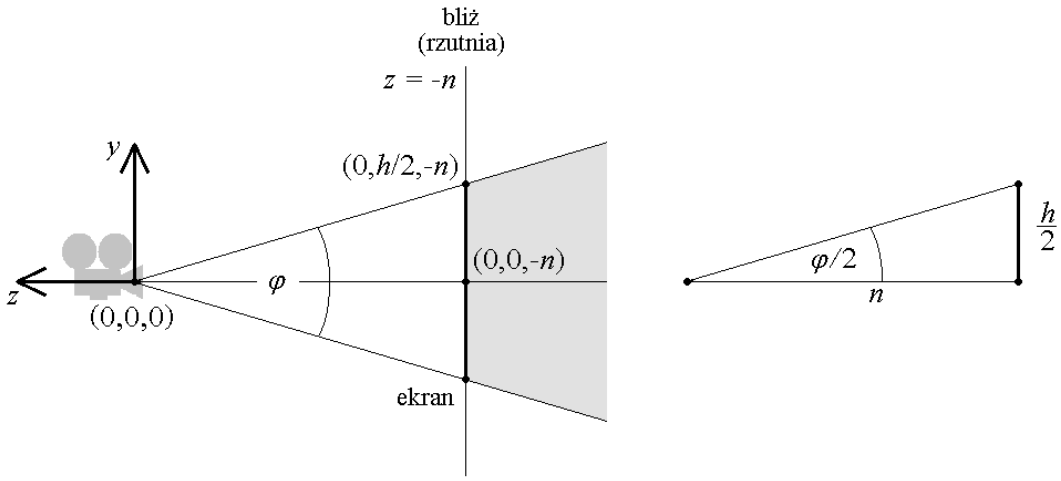
$$\hat{P} = \begin{pmatrix} \frac{2n}{w} & 0 & 0 & 0 \\ 0 & \frac{2n}{h} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Ponadto dla $w = 2$, $h = 2$ i $n = 1$ otrzymamy

(8.24)

$$\hat{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{f+1}{f-1} & -\frac{2f}{f-1} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Funkcje glFrustum vs gluPerspective



Rysunek 8.5. Widziany z boku (z kierunku +*x*) obszar przed ekranem

Biblioteka pomocnicza GLU (od ang. *OpenGL Utility Library*) zawiera często używaną funkcję `gluPerspective`. Efektem jej wywołania jest, podobnie jak w przypadku `glFrustum`, pomnożenie bieżącej macierzy przez macierz rzutowania perspektywicznego. Różni się jednak przyjmowanymi argumentami:

```
gluPerspective(φ, w/h, n, f);
```

Dwa ostatnie parametry *n* i *f* są tak, jak wcześniej odległością bliży i dali od kamery, drugi argument to stosunek szerokości *w* do wysokości *h* viewportu (ang. *aspect ratio*, my w zmiennej `wsp` przechowujemy stosunek wysokości do szerokości *h/w*, czyli odwrotność drugiego argumentu), a φ to kąt pionowego pola widzenia wyrażony w stopniach (ang. *fovy* od *field of view y*). Pokazuje to rysunek 8.5. Z tego rysunku wynika, że kąt ów można związać z rozmiarem ekranu i odległością ekranu od kamery korzystając z funkcji trygonometrycznych[2]. Weźmy pod uwagę górny trójkąt z rysunku rozciągnięty między trzema punktami o zaznaczonych na rysunku współrzędnych (dla wygody narysowany także z prawej strony). Jego wysokość równa jest połowie wysokości ekranu *h*/2, a długość podstawy równa odległości kamery od ekranu *n*. Zatem:

(8.25)

$$\operatorname{tg}\left(\frac{\varphi}{2}\right) = \frac{h/2}{n}$$

Argumentów metody `gluPerspective` jest mniej i są bardziej intuicyjne. Proszę jednak zauważyć, że możliwe jest to dzięki założeniu, że scena jest ustawiona symetrycznie względem kamery (jak we wzorze (8.23)). Założenie symetrii sceny oznacza, że samodzielnie implementując funkcję z takimi parametrami, jak w `gluPerspective`, możemy wykorzystać funkcję odpowiadającą `glFrustum`, ale nie odwrotnie.

Znajdźmy wartości parametrów funkcji `glFrustum` znając wartości parametrów funkcji `gluPerspective`. Z założenia symetrii wynika, że

(8.26)

$$l = -\frac{w}{2}, r = \frac{w}{2}, b = -\frac{h}{2}, t = \frac{h}{2}.$$

Ponadto możemy obliczyć wysokość ekranu korzystając ze wzoru (8.25), czyli:

(8.27)

$$h = 2n \operatorname{tg}\left(\frac{\varphi}{2}\right)$$

[2] Pamiętajmy tylko podczas implementacji, że funkcje trygonometryczne w C++ przyjmują jako argumenty kąty mierzone w radianach. Kąt pełny w radianach równy jest π, podczas gdy w stopniach to 180°.

Z kolei znając wysokość ekranu i jego proporcję, możemy łatwo znaleźć także jego szerokość, która równa jest iloczynowi tych dwóch wartości. Nie jestem jednak przekonany, czy wykorzystanie funkcji `glFrustum` jest opłacalne, czy nie łatwiej użyć bezpośrednio wzoru (8.23).

Odtworzenie macierzy identycznej, jak tej z rozdziału 1., ale używając metody `gluPerspective` wymagałoby użycia argumentów o wartościach widocznych w poniższym wywołaniu funkcji[3]:

```
gluPerspective(70.75f, 1/wsp, 1.0f, 10.0f);
```

gdzie $\varphi = 2arc\ tg(h/2n)$. A ponieważ wysokość ekranu równa była *h* = `wsp`·2 dla `wsp` równego 0.71, to $\varphi = 2arc\ tg(0.71) = 70.75°$.

Współrzędne viewportu

Układ współrzędnych NDC nie jest jeszcze ostatnim układem, jaki jest stosowany w grafice 3D. Na końcu potoku renderowania wykonywana jest jeszcze jedna liniowa transformacja: zmienne *x* i *y* są przeskalowywane w taki sposób, aby obejmowały zakres obszaru klienta, w którym znajduje się *viewport* i zaokrąglane do liczb całkowitych (jednostką jest teraz piksel), a głębokość przekształcana jest do zakresu od 0 do 1:

(8.28)

$$x_v = \frac{1}{2}(x_{NDC} + 1) \cdot W + X,$$

$$y_v = \frac{1}{2}(y_{NDC} + 1) \cdot H + Y,$$

$$z_v = \frac{1}{2}(z_{NDC} + 1).$$

To układ współrzędnych viewportu (ang. *viewport coordinates*). Wielkości *X* i *Y* wyznaczają lewy dolny róg *viewport*u, a *W* i *H* to odpowiednio jego szerokość i wysokość (we współrzędnych obszaru klienta okna). Zakres zmiennej *z* w układzie okna można kontrolować za pomocą funkcji `glDepthRange`, domyślnie jest to właśnie zakres od 0 do 1. Korzystając ze współrzędnych z_v (skonwertowanych do liczb całkowitych) przeprowadzany jest test głębi, a współrzędne x_v i y_v używane są do zapełnienia bufora ramki, w którym przygotowywany jest obraz widoczny na ekranie monitora.

Przykład

Sprawdźmy jak ten ciąg przekształceń wygląda w praktyce korzystając z ustawień użytych w pierwszym rozdziale. Umieściliśmy wówczas na scenie trójkąt o wierzchołkach w punktach *A* = (–1, –1, 0, 1), *B* = (1, –1, 0, 1) i *C* = (0, 1, 0, 1) (por. rozdział 1, w szczególności listing 1.17 MOŻE TO NIE PIERWSZY). To są współrzędne w lokalnym układzie odniesienia. Załóżmy, że macierz świata jest jednostkowa, a macierz widoku taka, że obiekt odsunięty jest od kamery o 10. W układzie kamery punkty mają zatem współrzędne *A* = (–1, –1, –3, 1), *B* = (1, –1, –3, 1) i *C* = (0, 1, –3, 1). Ponieważ do ustawienia macierzy rzutowania użyliśmy funkcji `glFrustum` z argumentami

```
glFrustum(-1.0f, 1.0f, wsp*-1.0f, wsp*1.0f, 1.0f, 10.0f);
```

to stałe określające bryłę widzenia równe są *l* = –1, *r* = 1, *n* = 1 i *f* = 10. Wartości *b* i *t* zależą od rozmiaru okna tak, aby proporcja wirtualnego ekranu na rzutni była taka sama, jak proporcja viewportu, a tym samym trójkąt widoczny na ekranie miał takie same kąty, co ten wirtualny. Dla okna o rozmiarze 800×600 w Windows 7, obszar klienta, którego całą powierzchnie zajmuje viewport ma wielkość *W*×*H* = 782×555. To oznacza, że proporcja `wsp` równa H/W będzie miała wartość 0.709718645 (jak zaznaczyłem wcześniej to jej odwrotność nazywana jest *aspect ratio*). Na potrzeby tego ćwiczenia zaokrąglimy ją do 0.71, co jest pewnie zbyt zgrubne biorąc pod uwagę fakt, że piksel może mieć szerokość mniejszą niż jedną tysięczną szerokości viewportu. Wobec tego *b* = –0.71, a *t* = 0.71. Macierz rzutowania perspektywicznego będzie zatem miała postać:

(8.29)

[3] Pamiętajmy tylko, że aby użyć metody `gluPerspective` musimy do kodu włączyć plik *GL\glu.h* (dyrektywa `#include <gl\GLU.h>`), a w ustawieniach linkera dołączyć bibliotekę *glu32.lib*.

$$\hat{P} = \begin{pmatrix} \frac{2 \cdot 1}{1-(-1)} & 0 & \frac{1+(-1)}{1-(-1)} & 0 \\ 0 & \frac{2 \cdot 1}{0.71-(-0.71)} & \frac{0.71+(-0.71)}{0.71-(-0.71)} & 0 \\ 0 & 0 & -\frac{10+1}{10-1} & -\frac{2 \cdot 1 \cdot 10}{10-1} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{2}{2} & 0 & \frac{0}{2} & 0 \\ 0 & \frac{2}{1.42} & \frac{0}{1.42} & 0 \\ 0 & 0 & -\frac{11}{9} & -\frac{20}{9} \\ 0 & 0 & -1 & 0 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & -1.22 & -2.22 \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Transformacja punktów A , B i C do współrzędnych przycinania wygląda następująco:

(8.30)

$$A_c = \hat{P}A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & -1.22 & -2.22 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ -1 \\ -3 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1.41 \\ 3.66-2.22 \\ 3 \end{pmatrix} = \begin{pmatrix} -1 \\ -1.41 \\ 1.44 \\ 3 \end{pmatrix},$$

$$B_c = \hat{P}B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & -1.22 & -2.22 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ -3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1.41 \\ 3.66-2.22 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1.41 \\ 1.44 \\ 3 \end{pmatrix},$$

$$C_c = \hat{P}C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & -1.22 & -2.22 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -3 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.41 \\ 3.66-2.22 \\ 3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.41 \\ 1.44 \\ 3 \end{pmatrix}.$$

Ponieważ wszystkie trzy punkty leżą na jednej płaszczyźnie prostopadłej do osi optycznej kamery $z_e = -3$, to werteksy różnią się tylko współrzędnymi x i y. Tak jest także w układzie przycięcia. To się nie zmieni po przejściu do układu NDC, czyli po podzieleniu przez $w_c = -z_e = 3$. Za to przekształcenie nie jesteśmy już odpowiedzialni pisząc shadery – wykonuje je karta graficzna. Gdyby trójkąt znajdował się jeszcze dalej od płaszczyzny bliży, np. w $z_e = -5.5$ (środek między bliżą a dalą), to współrzędna z_c wzrosłaby do 4.49. W układzie przycięcia zmienia się bowiem kierunek osi z. Dla $z_e = -10$ tj. gdyby trójkąt znajdował się w płaszczyźnie dali, otrzymujemy $z_c = 10$. W układzie NDC współrzędne punktów będą równe:

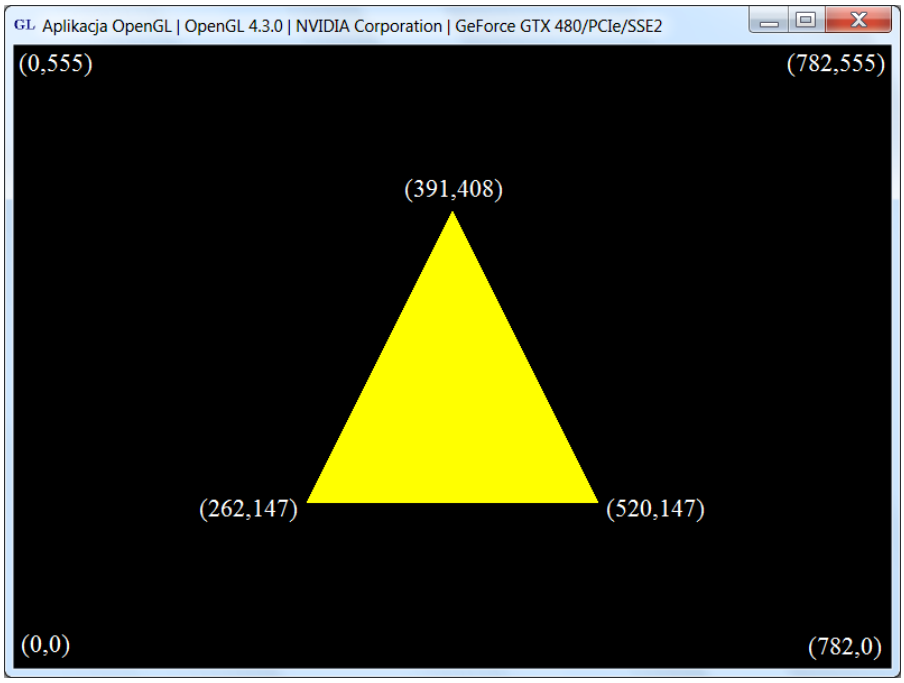
(8.31)

$$A_{NDC} = \begin{pmatrix} -1/3 \\ -1.41/3 \\ 1.44/3 \end{pmatrix} = \begin{pmatrix} -0.33 \\ -0.47 \\ 0.48 \end{pmatrix},$$

$$B_{NDC} = \begin{pmatrix} 1/3 \\ -1.41/3 \\ 1.44/3 \end{pmatrix} = \begin{pmatrix} 0.33 \\ -0.47 \\ 0.48 \end{pmatrix},$$

$$C_{NDC} = \begin{pmatrix} 0/3 \\ 1.41/3 \\ 1.44/3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.47 \\ 0.48 \end{pmatrix}.$$

Wartości współrzędnych x_c i y_c punktów A_{NDC} i B_{NDC} wskazują, że trójkąt powinien zajmować mniej więcej jedną trzecią szerokości okna i prawie połowę wysokości. I tak jest w rzeczywistości (por. rysunek 8.6).



Rysunek 8.6. Ostateczne współrzędne pikseli odpowiadających punktom A, B i C

A jak zmieniałyby się współrzędna z_{NDC}? Sprawdźmy to dla wartości, dla których przed chwilą obliczaliśmy współrzędną przycinania z_c. Dla $z_e = -5.5$ otrzymaliśmy $z_c = 4.49$, czyli $z_{NDC} = 4.49/5.5 = 0.81(63)$. Dla $z_e = 10$, $z_c = 10$, więc, jak należało się spodziewać $z_{NCD} = 1$.

Ostatnim przekształceniem, za które także nie jesteśmy już odpowiedzialni, jest przejście do układu współrzędnych viewportu (kontrolują je funkcje `glDepthRange` i `glViewport`). A ponieważ $X = 0$ i $Y = 0$ (nasz *viewport* zajmuje cały obszar klienta), to dla $W = 782$ i $H = 555$ przekształcenia te dane są wzorami:

(8.32)

$$x_v = W(x_{NDC} + 1)/2 = 391(x_{NDC} + 1),$$
$$y_v = H(y_{NDC} + 1)/2 = 277.5(y_{NDC} + 1),$$
$$z_v = (z_{NDC} + 1)/2.$$

W efekcie otrzymujemy (por. rysunek 8.6)

(8.33)

$$A_v = \begin{pmatrix} 262 \\ 147 \\ 0.74 \end{pmatrix},$$
$$B_v = \begin{pmatrix} 520 \\ 147 \\ 0.74 \end{pmatrix},$$
$$C_v = \begin{pmatrix} 391 \\ 408 \\ 0.74 \end{pmatrix}.$$

Sprawdźmy jak zmieni się obraz trójkąta, jeżeli rzut perspektywiczny zastąpimy rzutem izometrycznym, a więc jeżeli zamiast funkcji `glFrustum`, użyjemy funkcji `glOrtho` z tymi samymi argumentami:

```
glOrtho(-1.0f, 1.0f, wsp*-1.0f, wsp*1.0f, 1.0f, 10.0f);
```

dla `wsp` równego 0.709718645 ≈ 0.71. Wówczas macierz rzutowania przyjmie postać:

(8.34)

$$\hat{O} = \begin{pmatrix} \frac{2}{1-(-1)} & 0 & 0 & -\frac{1+(-1)}{1-(-1)} \\ 0 & \frac{2}{0.71-(-0.71)} & 0 & -\frac{1+(-1)}{0.71-(-0.71)} \\ 0 & 0 & -\frac{2}{10-1} & -\frac{10+1}{10-1} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{2}{2} & 0 & 0 & -\frac{0}{2} \\ 0 & \frac{2}{1.42} & 0 & -\frac{0}{1.42} \\ 0 & 0 & -\frac{2}{9} & -\frac{11}{9} \\ 0 & 0 & 0 & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & -0.22 & -1.22 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Korzystając z tej macierzy możemy przetransformować punkty A_e, B_e i C_e z układu kamery do układu przycinania:

(8.35)

$$A_c = \hat{O}A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & -0.22 & -1.22 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ -1 \\ -3 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1.41 \\ 0.66 - 1.22 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1.41 \\ -0.56 \\ 1 \end{pmatrix},$$

$$B_c = \hat{O}B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & -0.22 & -1.22 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ -3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1.41 \\ 0.66 - 1.22 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1.41 \\ -0.56 \\ 1 \end{pmatrix},$$

$$C_c = \hat{O}C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & -0.22 & -1.22 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -3 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.41 \\ 0.66 - 1.22 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.41 \\ -0.56 \\ 1 \end{pmatrix}.$$

W tym przypadku odległość werteksów od ekranu w żaden sposób nie wpływa na uzyskany obraz (współrzędne x_c i y_c), ma jedynie znaczenie przy teście głębi.

Konwersja do układu NDC jest trywialna, bo $w_c = 1$:

(8.36)

$$A_{NDC} = \begin{pmatrix} -1/1 \\ -1.41/1 \\ -0.56/1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1.41 \\ -0.56 \end{pmatrix},$$

$$B_{NDC} = \begin{pmatrix} 1/1 \\ -1.41/1 \\ -0.56/1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1.41 \\ -0.56 \end{pmatrix},$$

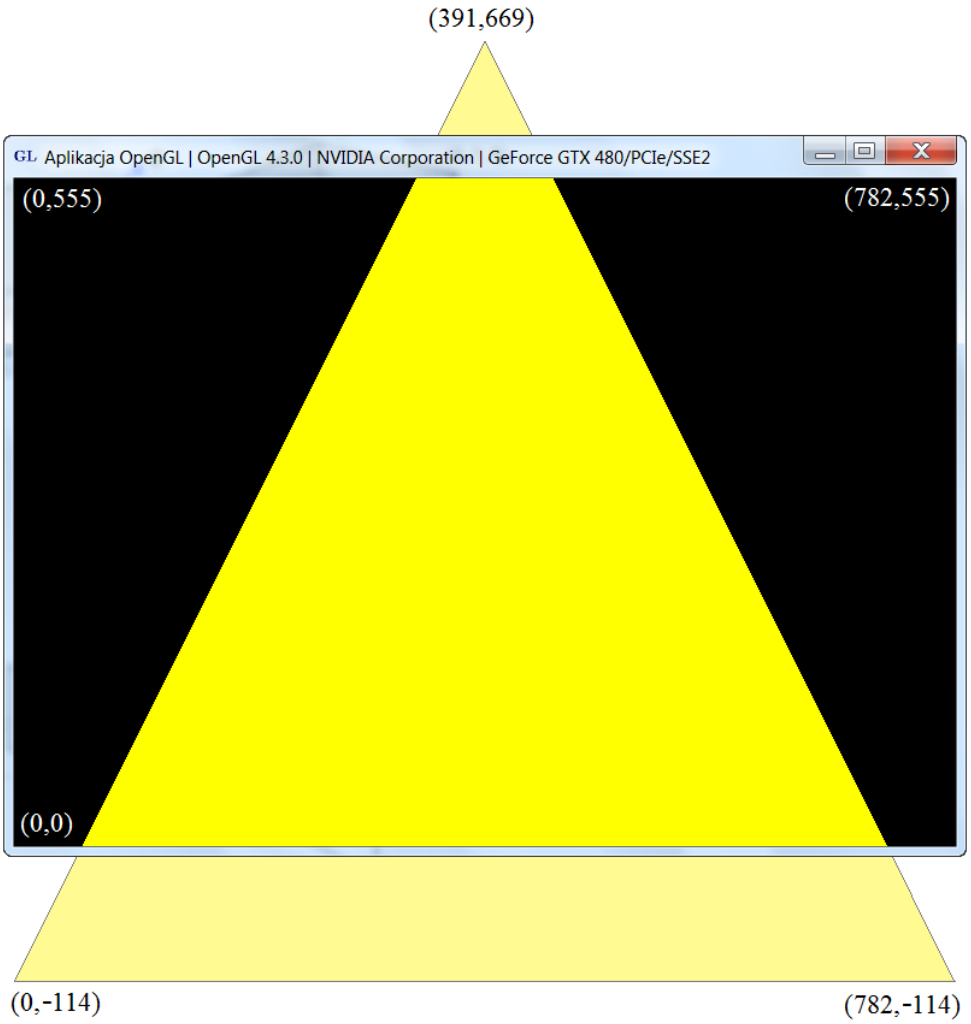
$$C_{NDC} = \begin{pmatrix} 0/1 \\ 1.41/1 \\ -0.56/1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.41 \\ -0.56 \end{pmatrix}.$$

Zwróćmy uwagę, że współrzędne y wszystkich punktów wykraczają poza zakres (–1, 1). To oznacza, że wierzchołki trójkąta znajdą się poza ekranem (rysunek 8.7). I nic tu nie zmieni zwiększanie odsunięcia kamery od ekranu. Sprawdźmy jeszcze jak zmieniałaby się współrzędna $z_c = z_{NDC}$, gdybyśmy przesunęli trójkąt w głąb do $z_e = -5.5$ i $z_e = -10$. Podstawiając te wartości do wzoru

(8.37)

$$z_{NDC} = z_c = -\frac{2}{9}z_e - \frac{11}{9}$$

otrzymamy odpowiednio 0 i 1. W odróżnieniu od przypadku rzutu perspektywicznego, w rzucie izometrycznym transformacja współrzędnej z jest liniowa.



Rysunek 8.7. Obraz w przypadku rzutowania izometrycznego. Część trójkąta nie mieści się w obrębie ekranu

I wreszcie przejście do współrzędnych viewportu (wzory 8.32) daje nam współrzędne pikseli odpowiadających wierzchołkom trójkąta widocznego na ekranie:

(8.38)

$$A_v = \begin{pmatrix} 0 \\ -114 \\ 0.22 \end{pmatrix},$$

$$B_v = \begin{pmatrix} 782 \\ -114 \\ 0.22 \end{pmatrix},$$

$$C_v = \begin{pmatrix} 391 \\ 669 \\ 0.22 \end{pmatrix}.$$

Czyli rzeczywiście współrzędne *y* dolnych wierzchołków są mniejsze od zera, a górnego większa od wysokości viewportu tj. 555 (rysunek 8.7).[4]

[4] Dodatkowe informacje i komentarze można znaleźć na stronach: *http://www.songho.ca/opengl/gl_projectionmatrix.html*, *http://www.scratchapixel.com/lessons/3d-advanced-lessons/perspective-and-orthographic-projection-matrix/perspective-projection-matrix/*, *http://www.opengl.org/wiki/Vertex_Transformation*, *http://mst.mimuw.edu.pl/lecture.php?lecture=gk1&part=Ch5*.

Macierz świata

Macierz świata odpowiada za transformacje współrzędnych lokalnego układu współrzędnych modelu do układu sceny. Układ ten może być dowolnie zorientowany – zwykle związany jest z trwałym podłożem widocznym na scenie, jeżeli takie jest obecne. Z tego powodu nie ma żadnych specjalnych funkcji, które by pozwalały taką macierz zbudować. Będzie to natomiast dla nas okazją, aby przyjrzeć się macierzom podstawowych przekształceń: translacji, skalowania, odbicia, pochylenia i obrotu. Poza pierwszym, wszystkie te przekształcenia mogą być zapisane za pomocą macierzy 3×3. Tylko translacja wymaga macierzy we współrzędnych jednorodnych. I od niej właśnie zaczniemy.

Translacja

Ciekawą własnością współrzędnych jednorodnych jest to, że umożliwiają opisanie przesunięcia o dowolny wektor za pomocą macierzy. W zwykłych współrzędnych kartezjańskich to nie jest możliwe – tam operacja przesunięcia opisywana jest przez dodanie wektorów:

(8.39)

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \vec{t} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} x + \Delta x \\ y + \Delta y \\ z + \Delta z \end{pmatrix}.$$

Analogiczny wynik, z tym, że dla czterowymiarowych wektorów, możemy uzyskać, jeżeli na współrzędne werteksu zadziałamy macierzą, w której czwarta kolumna zawierać będzie współrzędne wektora translacji:

(8.40)

$$\hat{T}\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x + \Delta x \cdot w \\ y + \Delta y \cdot w \\ z + \Delta z \cdot w \\ w \end{pmatrix}.$$

Zakładając, że $w = 1$, w wyniku działania macierzy $\hat{T}$ otrzymamy wektor przesunięty we współrzędnych x, y i z o wektor $\vec{t}$. Macierzą odwrotną do macierzy translacji o wektor $\vec{t}$ jest macierz translacji o wektor $-\vec{t}$.

W tradycyjnym OpenGL funkcja `glTranslate` mnoży bieżącą macierz przez macierz (8.39). Jej argumentami są wielkości Δx, Δy i Δz.

Skalowanie i odbicia względem płaszczyzn układu współrzędnych

Najprostszym przekształceniem jest skalowanie. Opisuje je macierz:

(8.41)

$$\hat{S} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Działając nią na dowolny wektor uzyskamy wektor, w którym poszczególne współrzędne będą przemnożone przez odpowiednie współczynniki s_x, s_y lub s_z:

(8.42)

$$\hat{S}\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \\ s_z z \\ w \end{pmatrix}.$$

W efekcie bryła, której macierz świata zawiera takie skalowanie zostanie w kierunku x powiększona s_x razy. Analogicznie w kierunkach y i z. Jeżeli wartości współczynników są mniejsze od jedności (ale dodatnie), bryła zostanie zmniejszona. Dla ujemnych wartości uzyskamy odbicia względem płaszczyzn wyznaczonych przez osie bieżącego układu współrzędnych. Dla $s_x < 0$ będzie to odbicie względem płaszczyzny OYZ.

Wielkości s_x, s_y i s_z są argumentami funkcji `glScale` tradycyjnego OpenGL, która mnoży bieżącą macierz przez macierz (N2).

Macierz może również zawierać współczynnik s_w (zamiast jedynki w ostatnim wierszu). Wówczas, w efekcie dzielenia perspektywicznego wszystkie współrzędne wektora zostaną zmniejszone s_w razy.

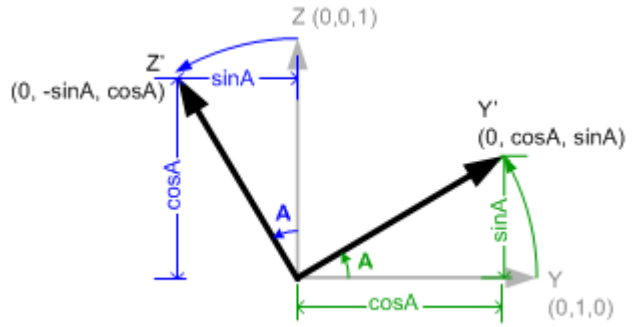
Obroty

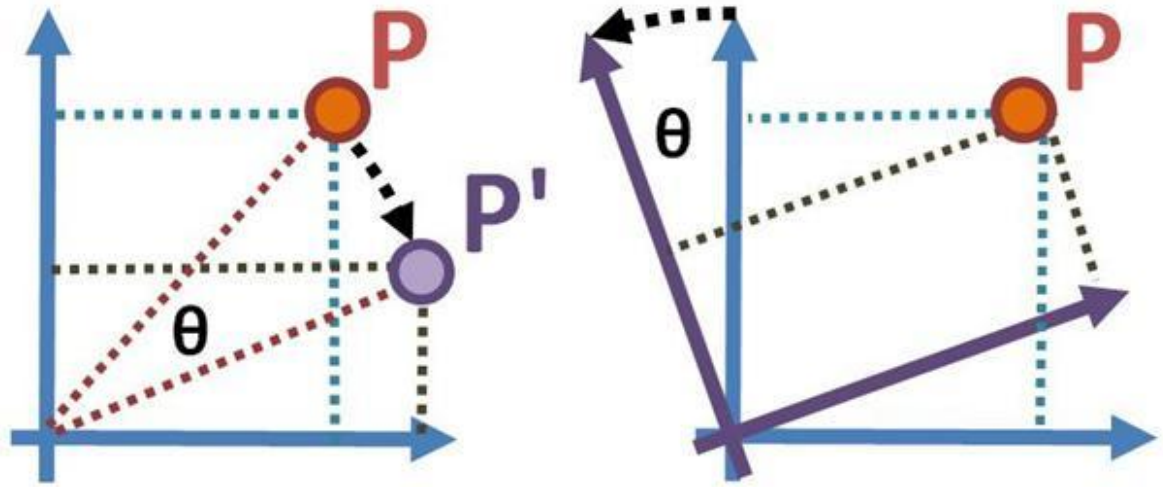
Wróćmy na chwilę do płaszczyzny, na której obroty są jednoznaczne. W takim przypadku jest bowiem tylko jedna możliwa oś obrotu – prostopadła do tej płaszczyzny. Macierz obrotu równa jest wówczas

(***)

$$\begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix},$$

gdzie φ to kąt obrotu wokół środka układu współrzędnych skierowany w kierunku przeciwnym do ruchu wskazówek zegara (rysunek ***).





DODAĆ TEŻ RYSUNEK DLA OSI

Ja nie obracam osi, tylko obiekt!!!!!!!!!!!!

W trzech wymiarach sprawa się komplikuje, bo oś obrotu możemy wybrać dowolnie. Pół biedy, jeżeli jest ona jedną z osi kartezjańskiego układu współrzędnych. Wówczas macierze obrotu przyjmują proste formy, które są prostym uogólnieniem wzoru ***. Najprościej to zobaczyć, jeżeli oś obrotu to będzie osią OZ. Wówczas zmiany następują w płaszczyźnie XY i macierz obrotu jest prostym uogólnieniem macierzy (***):

(***)

$$\hat{R}_z = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Sprawdźmy, jak macierz ta transformuje dowolny wektor określającym położenie punktu we współrzędnych jednorodnych:

$$\hat{R}_z \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} \cos(\gamma)x - \sin(\gamma)y \\ \sin(\gamma)x + \cos(\gamma)y \\ z \\ w \end{pmatrix}.$$

Zmieniają się zatem współrzędne punktu w taki sposób, że:

$$x' = \cos(\gamma)\, x - \sin(\gamma)\, y,$$
$$y' = \sin(\gamma)\, x + \cos(\gamma)\, y,$$
$$z' = z,$$
$$w' = w.$$

Sprawdźmy dla przykładu, jak obraca się punkt położony początkowo na osi OX: (1,0,0,1). W wyniku obrotu otrzymamy wektor o współrzędnych $(\cos(\gamma), \sin(\gamma), 0,1)$. To oznacza, że składowa x' maleje w miarę zwiększania kąta (jeżeli $\gamma < \pi$), a składowa y rośnie (dla $\gamma < \pi/2$). Zatem rzeczywiście punkt obraca się w stronę osi OY tj. w kierunku przeciwnym do wskazówek zegara.

Jeżeli założymy, że $w' = w = 1$, to możemy łatwo sprawdzić, że odległość obróconego punktu od punktu (0,0) nie zmienia się:

$$(x')^2 + (y')^2 + (z')^2 = (\cos(\gamma)\, x - \sin(\gamma)\, y)^2 + (\sin(\gamma)\, x + \cos(\gamma)\, y)^2 + z^2 =$$
$$= \cos^2(\gamma)x^2 - 2\cos(\gamma)\sin(\gamma)\, xy + \sin^2(\gamma)y^2 + \sin^2(\gamma)x^2 + 2\sin(\gamma)\cos(\gamma)\, xy + \cos^2(\gamma)y^2 + z^2 =$$
$$= \big(\cos^2(\gamma) + \sin^2(\gamma)\big)x^2 + \big(\sin^2(\gamma) + \cos^2(\gamma\varphi)\big)y^2 + z^2 = x^2 + y^2 + z^2.$$

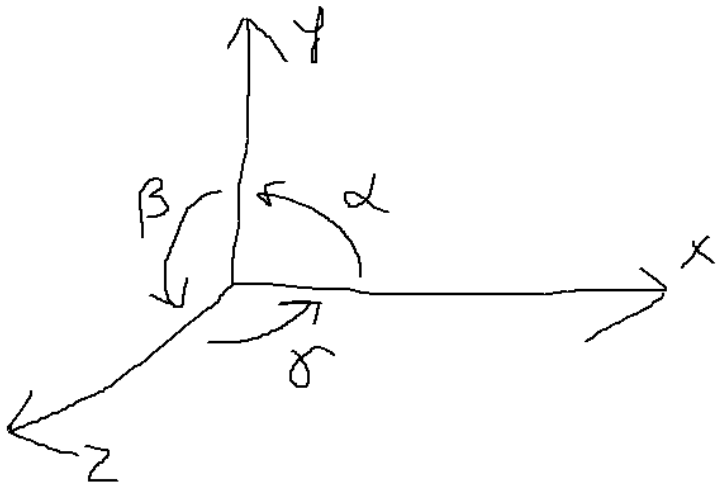
Analogicznie możemy zbudować macierz obrotu wokół osi OX. Wówczas zmianie ulegają tylko współrzędne *y* i *z*:

$$\hat{R}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

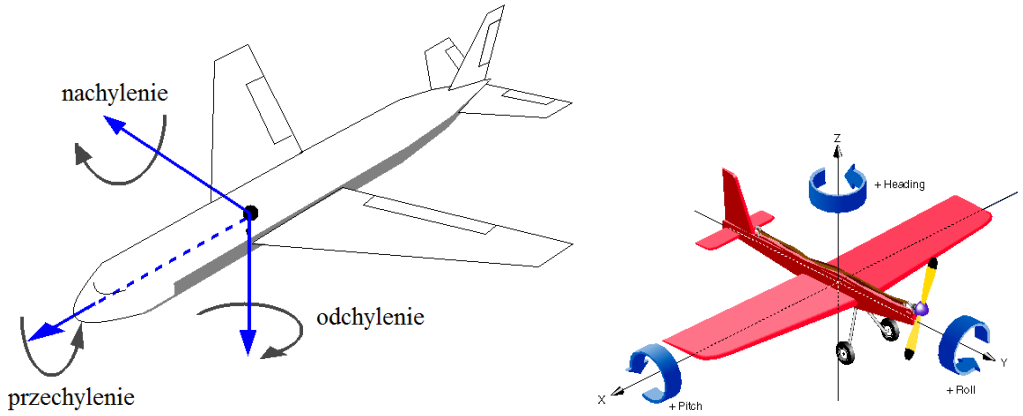
Nieco inaczej wygląda macierz obrotu wokół osi OY:

$$\hat{R}_y = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Przyczyna jest prosta. Zakładamy, że obrót punktu o dodatni kąt jest przeciwny do wskazówek zegara. W przypadku osi OZ oznacza to obrót punktu znajdującego się w pierwszej ćwiartce następuje od osi OX do osi OY. W przypadku osi OX – od osi OY do osi OZ. Natomiast w przypadku obrotu wokół osi OY, zmiana następuje od osi OZ do osi OX (rysunek **). Ta odwrócona kolejność osi powoduje, że macierz obrotu wygląda jak transponowana.



O obrotach możemy myśleć też w nieco inny sposób. Zamiast obracać punkty w układzie współrzędnych, możemy wyobrazić sobie, że przechodzimy do nowego układu współrzędnych o obróconych osiach. Osie nowego układu współrzędnych byłyby wówczas obrócone zgodnie ze wskazówkami zegara.



OŚ OY POWINNA BYĆ W GÓRĘ, A OX W PRAWO

Rysunek 4.1. Jak opisać orientację modelu?

Trzy kąty obrotu wokół osi układu współrzędnych związanego z obracanym obiektem, nazywane kątami Cardana, to typowy sposób opisu orientacji samolotów (rysunek 4.1). Kąt odpowiadający obrotom wokół osi pionowej nazywa się kurs, a jego zmiana to odchylenie (ang. odpowiednio *heading* i *yaw*). Obrót wokół osi skierowanej wzdłuż samolotu to przechylenie (*roll*), a wokół osi poziomej, prostopadłej do kierunku lotu to nachylenie (*pitch*). Te terminy stosowane są także w grafice 3D. Nachylenie (*pitch*) zmieniamy wówczas, gdy wolant ciągniemy do siebie lub pchamy od siebie. Wówczas samolot zadziera lub obniża nos. Jeżeli wolant przechylimy na bok, spowodujemy przechylenie samolotu (jedno skrzydło będzie wówczas wyżej od drugiego) – zmienia się kąt określany angielskim terminem *roll*, czyli właśnie przechylenie. Wreszcie naciśnięcie orczyka zmienia pozycję steru kierunkowego, a co za tym idzie, kurs samolotu, czyli *heading* lub *yaw*. Mylące w tym modelu jest to, że lekkie położenie na boku rzeczywistego samolotu w obecności grawitacji (tj. *roll*) również powoduje jego zmianę kierunku – dzieje się tak ze względu na zmianę kierunku siły nośnej, która jest prostopadła do płaszczyzny skrzydeł. Zamiast samolotu lepiej byłoby zatem wyobrażać sobie wahadłowiec w stanie nieważkości. Możemy też użyć jako przykładu własnej głowy. Możemy ją skręcać w lewo i prawo (oś obrotu wyznaczona jest przez kręgosłup), możemy spoglądać w dół i w górę (oś obrotu wyznaczona jest mniej więcej przez ostatni kręg kręgosłupa). Możemy także, choć w niezbyt dużym zakresie, przechylać głowę na boki.

Wróćmy jednak do samolotu. Ustalmy, że osie układu zorientowane są w taki sposób, że oś OZ wyznacza kierunek dziobu samolotu, a oś OY wskazuje jego górę. W konsekwencji oś OX przebiega wzdłuż jego prawego skrzydła. Wówczas kąt γ to przechylenie (*roll*), kąt β – nachylenie (*pitch*), a kąt α – odchylenie (*yaw*). Wszystkie trzy kąty mogą służyć do ustalenia zmiany orientacji samolotu. Ważna jest przy tym, należy to podkreślić kolejność, obrotów wokół poszczególnych osi. Najczęściej używana konwencja to rozpoczęcie od przechylenia, potem nachylenie i wreszcie odchylenie, a więc użycie macierzy: $\hat{R}_x\hat{R}_y\hat{R}_z$. Macierz, jaką wówczas uzyskamy to

$$\hat{R}_x\hat{R}_y\hat{R}_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ \sin(\alpha)\sin(\beta) & \cos(\alpha) & -\sin(\alpha)\cos(\beta) & 0 \\ -\cos(\beta)\sin(\beta) & \sin(\alpha) & \cos(\alpha)\cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos(\beta)\cos(\gamma) & -\cos(\beta)\sin(\gamma) & \sin(\beta) & 0 \\ \sin(\alpha)\sin(\beta)\cos(\gamma) + \cos(\alpha)\sin(\gamma) & -\sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & -\sin(\alpha)\cos(\beta) & 0 \\ -\cos(\alpha)\sin(\beta)\cos(\gamma) + \sin(\alpha)\sin(\gamma) & \cos(\alpha)\sin(\beta)\sin(\gamma) + \sin(\alpha)\cos(\gamma) & \cos(\alpha)\cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Używając tego wzoru do implementacji oszczędzimy na kilku mnożeniach.

W fizyce bryły sztywnej zamiast kątów Cardana używa się raczej kątów Eulera[5]. Różnią się tym od kątów Cardana, że jedna z osi obrotów wybierana spośród osi lokalnego układów współrzędnych zostanie powtórzona. Dzięki wcześniejszym obrotom, przy drugim obrocie nie jest ona już jednak tak samo zorientowana. Klasycznym zestawem jest kolejność osi: OZ, OX, obrócona oś OZ. Oznacza to macierz:

$$\hat{R}_z\hat{R}_x\hat{R}_z = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) & 0 & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos(\psi) & -\sin(\psi) & 0 & 0 \\ \sin(\psi) & \cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} \cos(\varphi) & -\sin(\varphi)\cos(\theta) & \sin(\varphi)\sin(\theta) & 0 \\ \sin(\varphi) & \cos(\varphi)\cos(\theta) & -\cos(\varphi)\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos(\psi) & -\sin(\psi) & 0 & 0 \\ \sin(\psi) & \cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

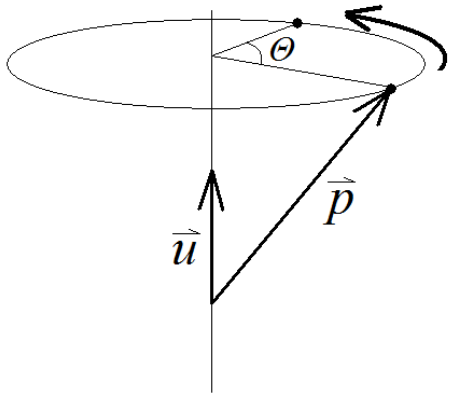
$$= \begin{pmatrix} \cos(\varphi)\cos(\psi) - \sin(\varphi)\cos(\theta)\sin(\psi) & -\cos(\varphi)\sin(\psi) - \sin(\varphi)\cos(\theta)\cos(\psi) & \sin(\varphi)\sin(\theta) & 0 \\ \sin(\varphi)\cos(\psi) + \cos(\varphi)\cos(\theta)\sin(\psi) & -\sin(\varphi)\sin(\psi) + \cos(\varphi)\cos(\theta)\cos(\psi) & -\cos(\varphi)\sin(\theta) & 0 \\ \sin(\theta)\sin(\psi) & \sin(\theta)\cos(\psi) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

W tym wypadku grozi nam jednak tak zwana blokada przegubowa. skopiować OPIS

Obrót wokół dowolnej osi

W praktycznych zastosowaniach bardzo ważna jest możliwość obracania modeli wokół wskazanej osi. To zadanie realizuje funkcja `glRotate` tradycyjnego OpenGL. Wyprowadzenie wzoru przeprowadzę we współrzędnych kartezjańskich, bo i bez współczynnika skalowania wzory będą bardzo długie. Załóżmy oś obrotu wyznaczoną przez jednostkowy werktor $\vec{u}$, przechodzącą przez początek kartezjańskiego układu współrzędnych $O = (0,0,0)$ i dowolny punkt $P = (x, y, z)$, który chcemy obrócić o θ przeciwnie do wskazówek zegara.

[5] Terminologia nie jest całkiem jednoznaczna. W inżynierii kąty Cardana nazywane są kątami Eulera. Często oba zestawy kątów, z powtórzeniem osi lub bez, nazywa się po prostu kątami Eulera.



Rysunek ***.*** Obrót punktu wokół osi

Zastanówmy się, w jaki sposób zapisać, że punkt *P* wskazywany przez wektor wodzący $\vec{p}$ został obrócony o kąt θ (rysunek 25.2)[6]. Jeżeli wektor $\vec{p}$ rozłożymy na część równoległą i prostopadłą do osi obrotu: $\vec{p} = \vec{p}_{\parallel} + \vec{p}_{\perp}$, to część równoległa nie ulegnie żadnej zmianie, natomiast część prostopadła będzie podlegać prawom obrotu dwuwymiarowego. W efekcie jeżeli obrócony punkt wskazywany jest przez wektor $\vec{p}' = \vec{p}'_{\parallel} + \vec{p}'_{\perp}$ to:

(25.22)

$$\vec{p}'_{\parallel} = \vec{p}_{\parallel}$$
$$\vec{p}'_{\perp} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \vec{p}_{\perp}$$

Dwuwymiarowa przestrzeń obrotu prostopadłej części wektora, w której zapisana została macierz w drugim wzorze, wyznaczona jest przez wektor $\vec{p}_{\perp}$ i wektor do niego prostopadły, a jednocześnie prostopadły do wektorów $\vec{u}$ i $\vec{p}_{\parallel}$. Można go zatem wyznaczyć, korzystając z iloczynu wektorowego $\vec{u} \times \vec{p}_{\perp}$. Załóżmy, że oś obrotu skierowana jest w kierunku *z*. Wówczas dwuwymiarowa przestrzeń prostopadła może być zapisana za pomocą współrzędnych *x* i *y*:

(25.23)

$$\vec{p}_{\perp} = \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$
$$\vec{u} \times \vec{p}_{\perp} = \begin{pmatrix} -p_y \\ p_x \end{pmatrix}$$

To oznacza, że:

(25.24)

$$\vec{p}'_{\perp} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \vec{p}_{\perp} = \begin{pmatrix} \cos(\theta)\, p_x - \sin(\theta) p_y \\ \sin(\theta)\, p_y + \cos(\theta) p_x \end{pmatrix} = \cos(\theta)\, \vec{p}_{\perp} + \sin(\theta)\, (\vec{u} \times \vec{p}_{\perp})$$

Ostatecznie obrócony wektor $\vec{p}'$ dany jest wzorem:

(25.25)

$$\vec{p}' = \vec{p}_{\parallel} + \cos(\theta)\, \vec{p}_{\perp} + \sin(\theta)\, (\vec{u} \times \vec{p}_{\perp})$$

Jak, korzystając z tego wzoru, napisać macierz obrotu, której elementy będą zawierały współrzędne wektora $\vec{u} = (u_x, u_y, u_z)$ i kąt θ? Przede wszystkim musimy przepisać wzór (25.25) w taki sposób, aby opisywał transformację całego wektora $\vec{p}$. Po pierwsze wykorzystajmy fakt, że iloczyn wektorowy wektorów $\vec{u}$ i $\vec{p}_{\parallel}$ jest równy zeru. Wobec tego $\vec{u} \times \vec{p}_{\perp} = \vec{u} \times \vec{p}$. Składową równoległą wektora $\vec{p}_{\parallel}$ rozłóżmy na dwie części

$$\vec{p}_{\parallel} = \cos(\theta)\, \vec{p}_{\parallel} + (1 - \cos(\theta))\, \vec{p}_{\parallel}.$$

Wówczas możemy przepisać wzór (25.25) jako:

$$\vec{p}' = \cos(\theta)\, \vec{p} + (1 - \cos(\theta))\, \vec{p}_{\parallel} + \sin(\theta)\, (\vec{u} \times \vec{p}) =$$

[6] Por. analogiczne rozumowanie w dodatku C.

$$= \cos(\theta)\,\vec{p} + (1-\cos(\theta))\,(\vec{p}\circ\vec{u})\vec{u} + \sin(\theta)\,(\vec{u}\times\vec{p}).$$

Przechodząc do zapisu macierzowego wykorzystajmy operator gwiazdki do zapisu iloczynu wektorowego oraz iloczyn tensorowy (diadyczny) do obliczenia składowej równoległej wektora (por. rozdział 2.). W efekcie uzyskamy[7]:

$$\vec{p}' = \{\cos(\theta)\,\hat{I} + (1-\cos(\theta))\,\vec{u}\otimes\vec{u} + \sin(\theta)\,(\vec{u}^{*})\}\vec{p} =$$

$$\vec{p}' = \hat{R}_{\vec{u}}\vec{p} = \left\{\cos(\theta)\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix} + (1-\cos(\theta))\begin{pmatrix}u_x^2&u_xu_y&u_xu_z\\u_xu_y&u_y^2&u_yu_z\\u_xu_z&u_yu_z&u_z^2\end{pmatrix} + \sin(\theta)\begin{pmatrix}0&-u_z&u_y\\u_z&0&-u_x\\-u_y&u_x&0\end{pmatrix}\right\}\vec{p}.$$

Uzyskaliśmy w ten sposób przekształcenie wektora $\vec{p}$, które, po zsumowaniu trzech wyrazów w nawiasie klamrowym, równoważne jest macierzy:

$$\hat{R}_{\vec{u}} = \begin{pmatrix}\cos(\theta)&0&0\\0&\cos(\theta)&0\\0&0&\cos(\theta)\end{pmatrix} + \begin{pmatrix}(1-\cos(\theta))\,u_x^2&(1-\cos(\theta))\,u_xu_y&(1-\cos(\theta))\,u_xu_z\\(1-\cos(\theta))\,u_xu_y&(1-\cos(\theta))\,u_y^2&(1-\cos(\theta))\,u_yu_z\\(1-\cos(\theta))\,u_xu_z&(1-\cos(\theta))\,u_yu_z&(1-\cos(\theta))\,u_z^2\end{pmatrix} +$$

$$+\begin{pmatrix}0&-\sin(\theta)\,u_z&\sin(\theta)\,u_y\\\sin(\theta)\,u_z&0&-\sin(\theta)\,u_x\\-\sin(\theta)\,u_y&\sin(\theta)\,u_x&0\end{pmatrix} =$$

$$= \begin{pmatrix}\cos(\theta)+(1-\cos(\theta))\,u_x^2&(1-\cos(\theta))\,u_xu_y-\sin(\theta)\,u_z&(1-\cos(\theta))\,u_zu_x+\sin(\theta)\,u_y\\(1-\cos(\theta))\,u_xu_y+\sin(\theta)\,u_z&\cos(\theta)+(1-\cos(\theta))\,u_y^2&(1-\cos(\theta))\,u_yu_z-\sin(\theta)\,u_x\\(1-\cos(\theta))\,u_zu_x-\sin(\theta)\,u_y&(1-\cos(\theta))\,u_yu_z+\sin(\theta)\,u_x&\cos(\theta)+(1-\cos(\theta))\,u_z^2\end{pmatrix}.$$

Sprawdźmy tę macierz, choćby dla jednego prostego przypadku. Załóżmy oś obrotu skierowaną wzdłuż osi OZ: $\vec{u} = (0,0,1)$. Wówczas macierz znacznie się upraszcza, bo wszystkie iloczyny różnych składowych wektora $\vec{u}$ są równe zeru i uzyskujemy macierz postaci:

$$\hat{R}_{(0,0,1)} =$$

$$= \begin{pmatrix}\cos(\theta)+(1-\cos(\theta))\,0&(1-\cos(\theta))\,0-\sin(\theta)\,1&(1-\cos(\theta))\,0+\sin(\theta)\,0\\(1-\cos(\theta))\,0+\sin(\theta)\,1&\cos(\theta)+(1-\cos(\theta))\,0&(1-\cos(\theta))\,0-\sin(\theta)\,0\\(1-\cos(\theta))\,0-\sin(\theta)\,0&(1-\cos(\theta))\,0+\sin(\theta)\,0&\cos(\theta)+(1-\cos(\theta))\,1\end{pmatrix} =$$

$$= \begin{pmatrix}\cos(\theta)&-\sin(\theta)&0\\\sin(\theta)&\cos(\theta)&0\\0&0&1\end{pmatrix} = \hat{R}_z$$

Uzyskaliśmy więc macierz ze wzoru (***).

I jeszcze ostatnia uwaga na temat obrotów. Rozmawiając o obrotach warto upewnić się, co nasz z

http://en.wikipedia.org/wiki/Active_and_passive_transformation

Macierze obrotu a kwaterniony jednostkowe

Poza kątami Cardana lub Eulera i macierzami, istnieje jeszcze trzeci sposób opisu obrotów – kwaterniony. Z ich pomocą można łatwo zapisać obrót wokół osi wyznaczonej przez wektor o zadany kąt. Można wykazać ich równoważność z macierzą (***). Więcej na ich temat w dodatku C. Warto także zaznaczyć, że kwaterniony pozwalają na zapisanie równań ruchu bryły sztywnej.

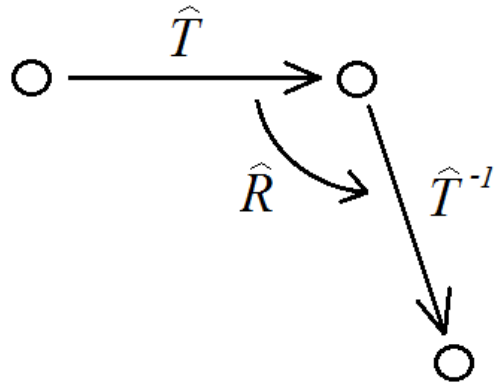
[7] Obrót wokół osi można również rozłożyć na pochylenie i skalowanie tak dobrane, aby rozmiar obracanej bryły się nie zmieniał.

Złożenie obrotów i translacji - obrót wokół dowolnego punktu

Wszystkie macierze obrotów, które wyżej przedstawiłem opisują obroty wokół środka bieżącego układu odniesienia. A co w przypadku, gdy chcemy wykonać obrót wokół innego punktu? W takim wypadku konieczne jest złożenie macierzy obrotu z macierzą translacji:

$$\hat{T}^{-1}(\vec{t})\hat{R}\hat{T}(\vec{t}) = \hat{T}(-\vec{t})\hat{R}\hat{T}(\vec{t}),$$

gdzie *R* to macierz obrotu, a *T* jest macierzą translacji o wektor $\vec{t}$. Wpierw przesuwamy się do punktu, wokół którego ma być wykonywany obrót, następnie wykonywany jest ów obrót. Wreszcie następuje przesunięcie z powrotem, ale w nowym, obróconym układzie (rysunek ***). Nie wracamy więc do tego samego punktu, a do punktu obróconego względem punktu do którego przesunęła nas macierz *T*.



Rysunek ****

Pochylenie

Mniej intuicyjnym przekształceniem, i stosunkowo rzadko używanym, jest pochylenie (ang. *shear* lub *skew*). Nie ma odpowiadającej mu funkcji w tradycyjnym OpenGL. Mówimy o nim w zasadzie tylko dlatego, że często jest niechcianym efektem błędnego zdefiniowania macierzy świata lub pojawia się w wyniku kumulacji błędów numerycznych. Pochylenie jest jedynym z omawianych w tej części przekształceń, które nie zachowuje kątów.

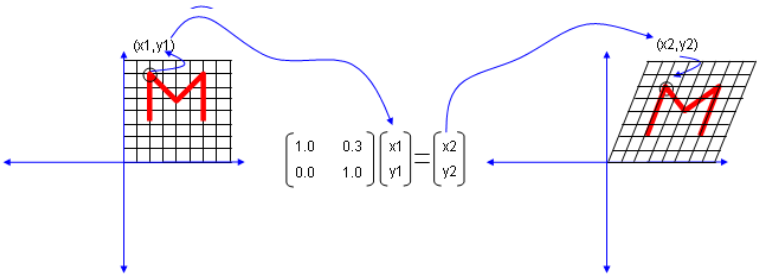
Zmodyfikujmy macierz jednostkową w taki sposób, aby poza jedynkami na diagonali, któryś z pozostałych elementów także był różny od zera. Oto przykład:

$$\begin{pmatrix} 1 & p_{xy} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Jeżeli współrzędne wektora przemnożymy przez taką macierz, uzyskamy:

$$\begin{pmatrix} 1 & p_{xy} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x + p_{xy}y \\ y \\ z \\ w \end{pmatrix}.$$

W efekcie współrzędne *x* zaczną zależeć od współrzędnej *y* sprzed transformacji $x' = x + p_{xy}x$ i bryły zostaną pochylone o kąt, którego tangensem jest wartość p_{yx} (rysunek 8.****).



Rysunek ***

Najłatwiej to zrozumieć myśląc o wersorach „zaszytych” w macierzy przekształcenia (zob. rozdział 2). Pomyślmy, że nowy układ współrzędnych nie ma już prostopadłych osi. Kąt między nimi zmalał bowiem o kąt równy $arctg(p_y)$. W efekcie zniekształceniu ulega cała scena.

Współrzędną x' można oczywiście pochylić poza płaszczyznę *OXY* uzależniając ją także od współrzędnej *z*. Pozwoli na to macierz:

$$\begin{pmatrix} 1 & p_{xy} & p_{xz} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

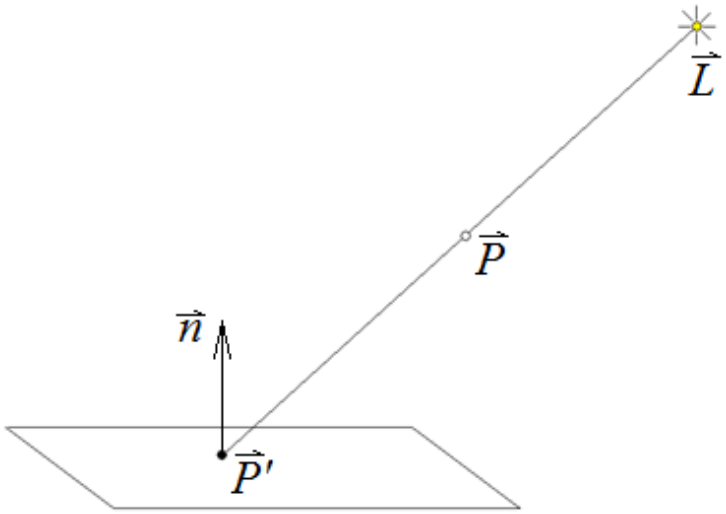
Analogicznie można wprowadzić pochylenie w osi *OY* lub *OZ*. Wystarczy tylko ustawić odpowiednie wartości elementów macierzy:

$$\begin{pmatrix} 1 & p_{xy} & p_{xz} & 0 \\ p_{yx} & 1 & p_{yz} & 0 \\ p_{zx} & p_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Pierwszy wiersz odpowiada za pochylenie osi *OX*, drugi *OY*, a trzeci *OZ*.

Rzut punku na płaszczyznę

Standardowe funkcje tradycyjnego OpenGL nie oferują wszystkich przydatnych transformacji. Dobrym przykładem jest macierz opisująca rzutowanie punktu na płaszczyznę (rozdział 2 i wzór (2.17)). Wykorzystuje się ją chociażby do prostego generowania cieni (rysunek 2.3). Należy jednak zwrócić uwagę, że macierz taka może być zapisana tylko we współrzędnych jednorodnych.



Rysunek B.1. Schemat rzutowania punktu na płaszczyznę

UZGODNIĆ OZNACZENIA

My jednak zaczniemy od prostych wzorów we współrzędnych kartezjańskich. Załóżmy, że płaszczyzna, na którą rzutujemy punkt (rzutnia), opisana jest wzorem:

$$\vec{p} \cdot \vec{n} = d$$

gdzie d to odległość płaszczyzny od początku układu współrzędnych, a $\vec{n}$ to trójwymiarowy wektor normalny do płaszczyzny. Rzutowany punkt $\vec{P}$ i środek rzutowania $\vec{L}$ (w przykładzie

generowania cienia środkiem rzutowania jest punktowe źródło światła) wyznaczają prostą (promień):

$$\vec{p} = \vec{P} + k(\vec{L} - \vec{P})$$

Na tej prostej musi oczywiście leżeć również rzut owego punktu $\vec{P}$ (rysunek B.1). Szukanym punktem jest więc przecięcie prostej i płaszczyzny. Wobec tego podstawmy wzór punktu na prostej do wzoru wyznaczającego punkty na płaszczyźnie:

$$\{\vec{P} + k(\vec{L} - \vec{P})\} \cdot \vec{n} = d$$

i wyznaczmy współczynnik *k*:

$$k = \frac{d - \vec{P} \cdot \vec{n}}{(\vec{L} - \vec{P}) \cdot \vec{n}}$$

Jeżeli wstawimy jawną postać tego współczynnika do wzoru na prostą, otrzymamy gotowy przepis na obliczenie rzutu punktu:

$$\vec{P'} = \vec{P} + \frac{d - \vec{P} \cdot \vec{n}}{(\vec{L} - \vec{P}) \cdot \vec{n}}(\vec{L} - \vec{P}) = \frac{\vec{P}\{(\vec{L} - \vec{P}) \cdot \vec{n}\} - (\vec{L} - \vec{P})(d - \vec{P} \cdot \vec{n})}{(\vec{L} - \vec{P}) \cdot \vec{n}}$$

$$= \frac{\vec{P}(\vec{L} \cdot \vec{n}) - \vec{P}(\vec{P} \cdot \vec{n}) + d(\vec{L} - \vec{P}) - \vec{L}(\vec{P} \cdot \vec{n}) + \vec{P}(\vec{P} \cdot \vec{n})}{(\vec{L} - \vec{P}) \cdot \vec{n}} =$$

$$= \frac{\vec{P}(\vec{L} \cdot \vec{n}) - \vec{L}(\vec{P} \cdot \vec{n}) + d(\vec{L} - \vec{P})}{(\vec{L} - \vec{P}) \cdot \vec{n}}$$

Dwa pierwsze wyrazy licznika tworzą podwójny iloczyn wektorowy. Ta informacja na niewiele nam się jednak przyda. Ważniejsze jest, że skróciły się dwa wyrazy, w których pojawiałyby się iloczyny składowych wektora $\vec{P}$. Nie oznacza to jeszcze, że położenie rzutu zależy liniowo od położenia rzutowanego punktu (co pozwalałoby na zapisanie rzutowania za pomocą macierzy). Współrzędne rzutowanego punktu $\vec{P}$ pojawiają się niestety w mianowniku powyższych wzorów:

$$P'_x = \frac{P_x(l_y n_y + l_z n_z - d) + P_y(-l_x n_y) + P_z(-l_x n_z) + d l_x}{l_x n_x + l_y n_y + l_z n_z - P_x n_x - P_y n_y - P_z n_z}$$

$$P'_y = \frac{P_x(-l_y n_x) + P_y(l_x n_x + l_z n_z - d) + P_z(-l_y n_z) + d l_y}{l_x n_x + l_y n_y + l_z n_z - P_x n_x - P_y n_y - P_z n_z}$$

$$P'_z = \frac{P_x(-l_z n_x) + P_y(-l_z n_y) + P_z(l_x n_x + l_y n_y - d) + d l_z}{l_x n_x + l_y n_y + l_z n_z - P_x n_x - P_y n_y - P_z n_z}$$

gdzie l_x, l_y i l_z to współrzędne wektora $\vec{L}$, a n_x, n_y i n_z – wektora $\vec{n}$. Wszystkie współrzędne rzutu $\vec{P'}$ skalowane są identycznym współczynnikiem o wartości $(\vec{L} - \vec{P}) \cdot \vec{n}$. To sugeruje, że aby pozbyć się mianownika, można przejść do współrzędnych jednorodnych, w których można go będzie przenieść do licznika współrzędnej *w* (przepis na mnożenie wektora we współrzędnych jednorodnych przez liczbę znajduje się w rozdziale 2 – wzór (2.15)). Po rozszerzeniu układu współrzędnych uzyskamy:

$$P'_x = P_x(l_y n_y + l_z n_z - d l_w) + P_y(-l_x n_y) + P_z(-l_x n_z) + P_w d l_x$$

$$P'_y = P_x(-l_y n_x) + P_y(l_x n_x + l_z n_z - d l_w) + P_z(-l_y n_z) + P_w d l_y$$

$$P'_z = P_x(-l_z n_x) + P_y(-l_z n_y) + P_z(l_x n_x + l_y n_y - d l_w) + P_w d l_z$$

$$P'_w = P_w(l_x n_x + l_y n_y + l_z n_z) + (-P_x n_x - P_y n_y - P_z n_z) l_w$$

Wektory $\vec{P}$ i $\vec{L}$ uzupełniamy o czwartą współrzędną *w*; możemy uznać, że w dotychczasowym rozumowaniu była ona już obecna, ale zawsze równa była jedności ($l_w = 1,\ P_w = 1$). Zgodnie z własnością współrzędnych jednorodnych wolne wyrazy opisujące translację o wektor $d\vec{L}$ znajdą się w czwartej kolumnie macierzy – wiążę je ze składową *w* wektora określającego położenie rzutowanego punktu. Zauważmy, że jeżeli wprowadzimy dodatkowo oznaczenie $n_w = -d$ i uporządkujemy wzory, to całe przekształcenie ujawni interesującą symetrię:

$$P'_x = P_x(n_y l_y + n_z l_z + n_w l_w) + P_y(-n_y l_x) + P_z(-n_z l_x) + P_w(-n_w l_x)$$

$$P'_y = P_x(-n_x l_y) + P_y(n_x l_x + n_z l_z + n_w l_w) + P_z(-n_z l_y) + P_w(-n_w l_y)$$

$$P'_z = P_x(-n_x l_z) + P_y(-n_y l_z) + P_z(n_x l_x + n_y l_y + n_w l_w) + P_w(-n_w l_z)$$

$$P'_w = P_x(-n_x l_w) + P_y(-n_y l_w) + P_z(-n_z l_w) + P_w(n_x l_x + n_y l_y + n_z l_z)$$

Nowe oznaczenie jest zgodne z definicją płaszczyzny we współrzędnych jednorodnych, która określona jest iloczynem skalarnym w tych współrzędnych:

$$P_x n_x + P_y n_y + P_z n_z + P_w n_w = 0$$

Nic już nie stoi na przeszkodzie, aby transformację punktu $\vec{P} = (P_x,\ P_y, P_z, P_w = 1)$ do jego rzutu $\vec{P}' = (P'_x, P'_y, P'_z, P'_w)$ zapisać za pomocą macierzy rzutowania perspektywicznego (por. wzór (2.17)):

$$\begin{pmatrix} \alpha - n_x l_x & -n_y l_x & -n_z l_x & -n_w l_x \\ -n_x l_y & \alpha - n_y l_y & -n_z l_y & -n_w l_y \\ -n_x l_z & -n_y l_z & \alpha - n_z l_z & -n_w l_z \\ -n_x l_w & -n_y l_w & -n_z l_w & \alpha - n_w l_w \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

Użyta w elementach diagonalnych liczba α równa jest iloczynowi skalarnemu we współrzędnych jednorodnych wektora określającego położenie źródła światła (lub ogólniej, środka rzutowania) i wektora normalnego do rzutni:

$$\alpha = n_x l_x + n_y l_y + n_z l_z + n_w l_w$$

Rozszerzenie współrzędnych kartezjańskich do współrzędnych jednorodnych, poza tym, że daje możliwość zapisu rzutowania za pomocą macierzy, ma także inną zaletę. W tych współrzędnych możemy przesunąć źródło światła do nieskończoności. Wprawdzie powyższe rozważania zakładają, że współrzędne *w* źródła światła i rzutowanego punktu są równe jedności ($l_w = 1,\ P_w = 1$), jednak wyprowadzona w ten sposób macierz rzutowania daje poprawne rezultaty również dla dowolnych wartości tych elementów, także gdy $l_w = 0$. Uzyskujemy wówczas rzut równoległy, który z tego punktu widzenia jest szczególnym przypadkiem rzutu perspektywicznego.

Macierz widoku

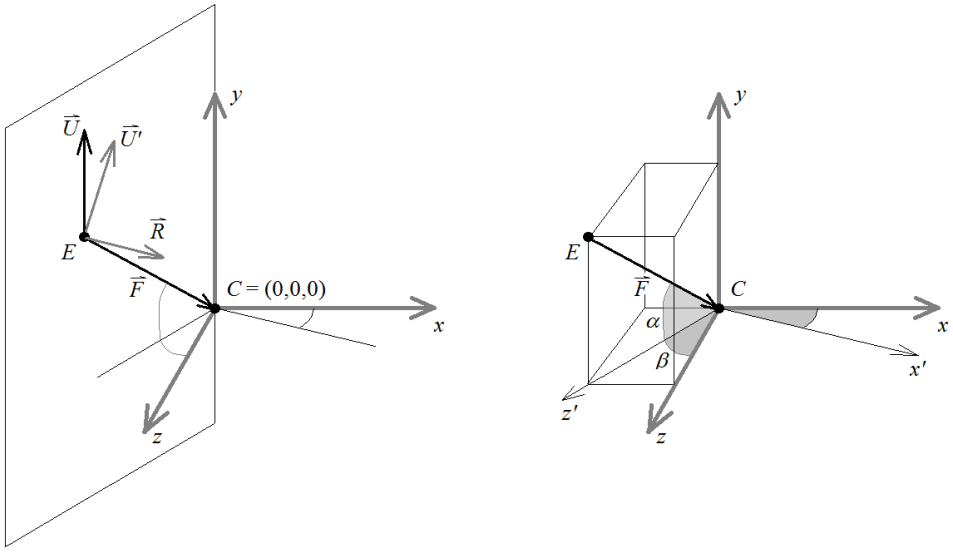
Macierz widoku określa transformację z układu sceny do układu kamery. Postaramy się odtworzyć macierz tworzoną przez wygodną funkcję `gluLookAt` z biblioteki GLU, której użyliśmy w pierwszym rozdziale do

określenia położenia i orientacji kamery. Jak pamiętamy, argumentami tej funkcji są trzy wektory określone w układzie współrzędnych sceny: wektor wskazujący położenie kamery *E*, punkt, na który kamera jest skierowana *C* i tzw. wektor polaryzacji $\vec{U}$ (symbol od ang. *up* – góra).

```
void gluLookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ,
               GLdouble centerX, GLdouble centerY, GLdouble centerZ,
               GLdouble upX, GLdouble upY, GLdouble upZ);
```

Punkty *E* i *C* wyznaczają wektor $\vec{F}$ (od ang. *forward* – do przodu) i zarazem oś optyczną kamery. Wzdłuż tej osi (od punkty *C* do *E*) skierowana będzie oś *OZ* w układzie współrzędnych kamery. Kamera skierowana na punkt *C* może być dowolnie obrócona wokół osi optycznej i dlatego konieczny jest wektor polaryzacji, który wyznacza jej orientację. Rysunek ***, który ilustruje tę sytuację nie pokazuje jednak sytuacji ogólnej, choć bardzo często spotykaną. Zakłada bowiem, że kamera skierowana jest na środek układu współrzędnych sceny, zatem *C* = *O* = (0,0,0), a wektor $\vec{U} = (0,1,0)$ jest skierowany wzdłuż osi *OY*. Z prawego rysunku widać, że macierz tworzoną przez funkcję `gluLookAt` można wyprowadzić przez złożenie dwóch obrotów: pierwszego wokół osi *OY* o kąt β, i drugiego wokół nowej obróconej osi *OX'* o kąt α, oraz translacji do punktu *E*, czyli w naszym szczególnym przypadku o wektor $-\vec{F}$ (por. listing 1.***). Łatwiej jednak znaleźć ją traktując ją jako transformację układu współrzędnych i pamiętając, że kolumny macierzy 3×3 stanowią wersory[8] nowego układu współrzędnych wyrażone w starym układzie współrzędnych, a zatem wersory osi *OX*, *OY* i *OZ* układu współrzędnych kamery widziane z układu współrzędnych związanego ze sceną. W czwartej kolumnie wstawimy położenie punktu *E*, dzięki czemu macierz będzie także uwzględniała przesunięcie początku układu współrzędnych do położenia kamery.

ZMIENIĆ L NA F (forward)



Rysunek ****. Położenie i orientacja kamery w układzie współrzędnych sceny

Po transformacji do układu współrzędnych kamery, oś *OZ* będzie wyznaczona przez wektor $-\vec{F}$ (skierowana od centrym do kamery). Oś *OX* będzie jednocześnie prostopadła do osi *OZ* i wektora $\vec{U}$ (wektory $\vec{U}$ i $\vec{F}$ nie mogą być równoległe!). Wskazywać będzie ją wektor $\vec{R}$ (od ang. *right*, czyli prawo). Wektor $\vec{U}$ jednoznacznie wyznacza polaryzację kamery, ale niekoniecznie jest prostopadła do wektorów $-\vec{F}$ i $\vec{R}$, dlatego należy znaleźć $\overrightarrow{U'}$, który ten warunek będzie spełniał, pozostając w płaszczyźnie napiętej przez wektory $-\vec{F}$ i $\vec{U}$ (zaznaczona na rysunku). To oznacza, że należy wykonać następującą ogólną konstrukcję:

1. oblicz wektor $\vec{F} = \vec{C} - \vec{E}$,
2. unormuj ten wektor i jego wartość zapisz do $\overrightarrow{F'} = \vec{F}/|\vec{F}|$,
3. oblicz wektor $\vec{R}$ prostopadły do wektorów $\overrightarrow{F'}$ i $\vec{U}$ korzystając z iloczynu wektorowego $\vec{R} = \overrightarrow{F'} \times \vec{U}$ (zwrot wyniku wyznacza reguła śruby prawoskrętnej),
4. unormuj wektor $\vec{R}$,

[8] Przypomnę, że wersory to wektory jednostkowe wskazujące kierunek osi współrzędnych układu.

5. oblicz wektor $\overrightarrow{U'} = \vec{R} \times \overrightarrow{F'}$ prostopadły do wektorów $\vec{R}$ i $\overrightarrow{F'}$ (wektor $\overrightarrow{U'}$ będzie jednostkowy, bo oba czynniki są jednostkowe, a jednocześnie prostopadłe do siebie).

Wersorami nowego układu współrzędnych kamery (wyrażonymi we współrzędnych sceny) są zatem: $\hat{x}' = \vec{R}$, $\hat{y}' = \overrightarrow{U'}$ i $\hat{z}' = -\overrightarrow{F'}$, co oznacza, że macierz widoku obracająca współrzędne z układu sceny do układu kamery (jeszcze bez przesunięcia) powinna wyglądać następująco CZEMU NIE KOLUMNY, BO TRANSPONOWANA:

$$\begin{pmatrix} R_x & R_y & R_z & 0 \\ U'_x & U'_y & U'_z & 0 \\ -F'_x & -F'_y & -F'_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

A co z przesunięciem? W rozdziale 1. sprawdziliśmy, że w przypadku, w którym *C* = *O* = (0,0,0) mamy dwie równoważne opcje: możemy zacząć serię przekształceń od przesunięcia kamery do punktu *E* tj. o wektor $-\vec{F}$, a potem wykonać obroty lub najpierw wykonać obroty, a dopiero po nich przesunąć scenę w kierunku przeciwnym do osi *OZ* w nowym układzie kamery o odległość równą długości wektora $\vec{F}$ tj. o wektor $-|\vec{F}|\hat{z}'$:

$$\hat{V} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -|\vec{F}| \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_x & R_y & R_z & 0 \\ U'_x & U'_y & U'_z & 0 \\ -F'_x & -F'_y & -F'_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} R_x & R_y & R_z & 0 \\ U'_x & U'_y & U'_z & 0 \\ -F'_x & -F'_y & -F'_z & -|\vec{F}| \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

W tym drugim przypadku macierz widoku wygląda znacznie lepiej, ale to pierwszy przypadek jest ogólniejszy i działa dla dowolnego położenia punktu C:

$$\hat{V} = \begin{pmatrix} R_x & R_y & R_z & 0 \\ U'_x & U'_y & U'_z & 0 \\ -F'_x & -F'_y & -F'_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -E_x \\ 0 & 1 & 0 & -E_y \\ 0 & 0 & 1 & -E_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} R_x & R_y & R_z & -\vec{R} \circ \vec{E} \\ U'_x & U'_y & U'_z & -\overrightarrow{U'} \circ \vec{E} \\ -F'_x & -F'_y & -F'_z & \overrightarrow{F'} \circ \vec{E} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

W przypadku, w którym *C* = *O* = (0,0,0), wektor $\vec{E} = -\vec{F}$, a ponadto jest prostopadły do wektorów $\vec{R}$ i $\overrightarrow{U'}$. W efekcie pierwsze dwa wyrazy czwartej kolumny równe są zeru, a trzeci staje się długością wektora $\vec{F}$ ze zmienionym znakiem.

Sprawdźmy jak konstrukcja macierzy widoku wygląda w praktyce. Najpierw rozważmy prosty, wręcz trywialny przykład, w którym kamera znajduje się w *E* = (0,0,3) i jest skierowana jest na punkt C = (0,0,0) (w układzie sceny). Wektor polaryzacji ustalmy równy $\vec{U} = [0,1,0]$. To przykład z pierwszego rozdziału, w którym scena i kamera są rozsunięte o 3 jednostki wzdłuż osi *OZ* (rysunek 1.***). Wówczas uzyskujemy:

1. $\vec{F} = (0,0,0) - (0,0,3) = [0,0,-3]$,
2. $|\vec{F}| = \sqrt{0^2 + 0^2 + (-3)^2} = \sqrt{9^2} = 3$, $\overrightarrow{F'} = \vec{F}/|\vec{F}| = \frac{[0,0,-3]}{3} = [0,0,-1]$,
3. $\vec{R} = \overrightarrow{F'} \times \vec{U} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ F'_x & F'_y & F'_z \\ U_x & U_y & U_z \end{vmatrix} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{vmatrix} =$
$= \hat{x}(0 \cdot 0 - (-1) \cdot 1) - \hat{y}(0 \cdot 0 - (-1) \cdot 0) + \hat{z}(0 \cdot 0 - 0 \cdot 1) = \hat{x} = [1,0,0]$,
4. $|\vec{R}| = \sqrt{1^2 + 0^2 + 0^2} = \frac{1}{\sqrt{2}}$, po unormowaniu: $\vec{R} = [1,0,0]$,
5. $\overrightarrow{U'} = \vec{R} \times \overrightarrow{F'} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ R_x & R_y & R_z \\ F'_x & F'_y & F'_z \end{vmatrix} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{vmatrix} =$
$= \hat{x}(0 \cdot (-1) - 0 \cdot 0) - \hat{y}(1 \cdot (-1) - 0 \cdot 0) + \hat{z}(1 \cdot 0 - 0 \cdot 0) = \hat{y} = [0,1,0]$.

Wektory $\vec{R} = [1,0,0]$, $\overrightarrow{U'} = [0,1,0]$ i $\overrightarrow{F'} = [0,0,-1]$ są skierowane odpowiednio wzdłuż osi *OX*, *OY* i przeciwnie do osi *OZ* w układzie kamery, są więc na pewno do siebie prostopadłe. W efekcie macierz widoku nie będzie obracała układu sceny, a będzie go jedynie przesuwać. Będzie to więc zwykła macierz translacji:

$$\hat{V} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

To oczywiście bardzo prosty przykład. Sprawdźmy więc, co się stanie, jeżeli przesuniemy kamerę do punktu *E* = (–1,1,1) nie zmieniając punktu *C* = (0,0,0), na który kamera jest skierowana. To bardziej odpowiada sytuacji z rysunku *** (por. też rysunki 1.*** i 1.*** (OBRAZ I SCHEMAT) z rozdziału 1.)

1. $\vec{F} = (0,0,0) - (-1,1,1) = [1,-1,-1]$,
2. $\left|\vec{F}\right| = \sqrt{1^2 + (-1)^2 + (-1)^2} = \sqrt{3}$, $\overrightarrow{F'} = \vec{F}/\left|\vec{F}\right| = \frac{[1,-1,-1]}{\sqrt{3}} = \left[\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right]$,
3. $\vec{R} = \overrightarrow{F'} \times \vec{U} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ F'_x & F'_y & F'_z \\ U_x & U_y & U_z \end{vmatrix} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ 0 & 1 & 0 \end{vmatrix} =$
$= \hat{x}\left(-\frac{1}{\sqrt{3}} \cdot 0 - \left(-\frac{1}{\sqrt{3}}\right) \cdot 1\right) - \hat{y}\left(\frac{1}{\sqrt{3}} \cdot 0 - \left(-\frac{1}{\sqrt{3}}\right) \cdot 0\right) + \hat{z}\left(\frac{1}{\sqrt{3}} \cdot 1 - \left(-\frac{1}{\sqrt{3}}\right) \cdot 0\right) = \left[\frac{1}{\sqrt{3}}, 0, \frac{1}{\sqrt{3}}\right]$,
4. $\left|\vec{R}\right| = \sqrt{\left(\frac{1}{\sqrt{3}}\right)^2 + 0^2 + \left(\frac{1}{\sqrt{3}}\right)^2} = \sqrt{\frac{2}{3}}$, a zatem po unormowaniu $\vec{R} = \left[\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right]$,
5. $\overrightarrow{U'} = \vec{R} \times \overrightarrow{F'} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ R_x & R_y & R_z \\ F'_x & F'_y & F'_z \end{vmatrix} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{vmatrix} =$
$= \hat{x}\left(0 \cdot \left(-\frac{1}{\sqrt{3}}\right) - \frac{1}{\sqrt{2}} \cdot \left(-\frac{1}{\sqrt{3}}\right)\right) - \hat{y}\left(\frac{1}{\sqrt{2}} \cdot \left(-\frac{1}{\sqrt{3}}\right) - \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{3}}\right) + \hat{z}\left(\frac{1}{\sqrt{2}} \cdot \left(-\frac{1}{\sqrt{3}}\right) - 0 \cdot \frac{1}{\sqrt{3}}\right) = \left[\frac{1}{\sqrt{6}}, \frac{2}{\sqrt{6}}, -\frac{1}{\sqrt{6}}\right]$.

Wektor $\overrightarrow{U'}$, choć nietrywialny, jest jednak jednostkowy, co łatwo sprawdzić obliczając jego normę:

$$\overrightarrow{U'} = \sqrt{\left(\frac{1}{\sqrt{6}}\right)^2 + \left(\frac{2}{\sqrt{6}}\right)^2 + \left(-\frac{1}{\sqrt{6}}\right)^2} = \sqrt{\frac{1}{6} + \frac{4}{6} + \frac{1}{6}} = 1$$

Sprawdźmy także, czy wektory $\vec{R} = \left[\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right]$, $\overrightarrow{U'} = \left[\frac{1}{\sqrt{6}}, \frac{2}{\sqrt{6}}, -\frac{1}{\sqrt{6}}\right]$ i $\overrightarrow{F'} = \left[\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right]$ są do siebie prostopadłe. Najłatwiej zrobić to obliczając ich iloczyny skalarne:

$$\vec{R} \circ \overrightarrow{U'} = \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{6}} + 0 \cdot \frac{2}{\sqrt{2}} + \frac{1}{\sqrt{2}} \cdot \left(-\frac{1}{\sqrt{6}}\right) = \frac{1}{\sqrt{12}} - \frac{1}{\sqrt{12}} = 0,$$

$$\vec{R} \circ \overrightarrow{F'} = \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{3}} + 0 \cdot \left(-\frac{1}{\sqrt{3}}\right) + \frac{1}{\sqrt{2}} \cdot \left(-\frac{1}{\sqrt{3}}\right) = \frac{1}{\sqrt{6}} - \frac{1}{\sqrt{6}} = 0,$$

$$\overrightarrow{U'} \circ \overrightarrow{F'} = \frac{1}{\sqrt{6}} \cdot \frac{1}{\sqrt{3}} + \frac{2}{\sqrt{6}} \cdot \left(-\frac{1}{\sqrt{3}}\right) + \left(-\frac{1}{\sqrt{6}}\right) \cdot \left(-\frac{1}{\sqrt{3}}\right) = \frac{1}{\sqrt{18}} - \frac{2}{\sqrt{18}} + \frac{1}{\sqrt{18}} = 0.$$

Łatwo się przekonać, że macierz, którą możemy dzięki tym wektorom skonstruować, a więc

$$\hat{V} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 0 \\ -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 0 \\ -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & -\sqrt{3} \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$\approx \begin{pmatrix} 0.71 & 0 & 0.71 & 0 \\ 0.41 & 0.82 & -0.41 & 0 \\ -0.58 & 0.58 & 0.58 & -1.73 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

jest ortonormalna, a ponadto użyta jako macierz model-widok (listing ***[9]) spowoduje, że trójkąt na ekranie będzie wyglądał tak samo, jak na rysunku ***.

[9] Ze względu na ułożenie elementów macierzy w tablicy kolumnami (por. wzór (2.***)), macierz w kodzie wydaje się transponowana.

Listing ***. **** (por. listingi 1.*** i 1.*** z rozdziału 1.)

```
void COknoGL::RysujScene()
{
    const float x0 = 1.0f;
    const float y0 = 1.0f;
    const float z0 = 1.0f;

    //Przygotowanie bufora
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //czyści bufory
    glLoadIdentity(); //macierz model-widok = macierz jednostkowa

    //Kamera w E=(-1,1,1) skierowana na C=(0,0,0), U=(0,1,0)
    /*
    glRotatef(35.26f, 1.0f, 0.0f, 0.0f); //3
    glRotatef(45.0f, 0.0f, 1.0f, 0.0f); //2
    glTranslatef(1, -1, -1); //1
    */

    /*
    gluLookAt(
      -1.0f, 1.0f, 1.0f,
       0.0f, 0.0f, 0.0f,
       0.0f, 1.0f, 0.0f);
    */

    float V[16] = {
        0.71f, 0.41f, -0.58f, 0.0f,
        0.0f, 0.82f, 0.58f, 0.0f,
        0.71f, -0.41f, 0.58f, 0.0f,
        0.0f, 0.0f, -1.73f, 1.0f
    };
    glMultMatrixf(V);

    //Żółty kolor werteksów
    glColor4f(1.0f, 1.0f, 0.0f, 1.0f);

    //Rysowanie trójkąta
    glBegin(GL_TRIANGLES);
    //ustalanie trzech wierzchołków trójkąta (werteksów (x,y,z))
    //(0,0,z) jest mniej więcej w środku ekranu
    glVertex3f(-x0, -y0, 0.0f); //dolny lewy
    glVertex3f(x0, -y0, 0.0f); //dolny prawy
    glVertex3f(0, y0, 0.0f); //górny
    //koniec rysowania figury
    glEnd();
```

```
        //Z bufora na ekran
        SwapBuffers(uchwytDC);
    }
```

Przykład złożenia macierzy widoku, świata i rzutowania

Sprawdźmy teraz, jak trójkąt o wierzchołkach A = (–1, –1, 0, 1), B = (1, –1, 0, 1) i C = (0, 1, 0, 1) z wcześniejszego przykładu będzie widoczny na ekranie, jeżeli macierzą widoku będzie macierz odpowiadająca kamerze ustawionej w punkcie E = (–1,1,1) skierowana na punkt O = (0,0,0) z wektorem polaryzacji równym $\vec{U} = [0,1,0]$ i zastosujemy rzutowanie perspektywiczne z parametrami ekranu takimi samymi, jak we wcześniejszym przykładzie. Macierz świata pozostanie macierzą jednostkową. Wiemy już, że macierz widoku w tym przypadku równa jest

$$\hat{V} \approx \begin{pmatrix} 0.71 & 0 & 0.71 & 0 \\ 0.41 & 0.82 & -0.41 & 0 \\ -0.58 & 0.58 & 0.58 & -1.73 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

a macierz rzutowania to

$$\hat{P} \approx \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & -1.22 & -2.22 \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Zatem iloczyn macierzy świata, widoku i rzutowania równa jest:

$$\hat{M}_{MVP} = \hat{P}\hat{V}\hat{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.41 & 0 & 0 \\ 0 & 0 & -1.22 & -2.22 \\ 0 & 0 & -1 & 0 \end{pmatrix}\begin{pmatrix} 0.71 & 0 & 0.71 & 0 \\ 0.41 & 0.82 & -0.41 & 0 \\ -0.58 & 0.58 & 0.58 & -1.73 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$\approx \begin{pmatrix} 0.71 & 0 & 0.71 & 0 \\ 0.58 & 1.16 & -0.58 & 0 \\ 0.71 & -0.71 & -0.71 & -0.11 \\ 0.58 & -0.58 & -0.58 & 1.73 \end{pmatrix}.$$

Zastosujmy tę macierz na współrzędnych punktów A, B i C (w układzie własnym modelu), aby uzyskać współrzędne tych punktów w układzie przycinania:

$$A_c = \hat{M}_{MVP}A = \begin{pmatrix} 0.71 & 0 & 0.71 & 0 \\ 0.58 & 1.16 & -0.58 & 0 \\ 0.71 & -0.71 & -0.71 & -0.11 \\ 0.58 & -0.58 & -0.58 & 1.73 \end{pmatrix}\begin{pmatrix} -1 \\ -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.71 \\ -1.75 \\ -0.11 \\ 1.73 \end{pmatrix},$$

$$B_c = \hat{M}_{MVP}B = \begin{pmatrix} 0.71 & 0 & 0.71 & 0 \\ 0.58 & 1.16 & -0.58 & 0 \\ 0.71 & -0.71 & -0.71 & -0.11 \\ 0.58 & -0.58 & -0.58 & 1.73 \end{pmatrix}\begin{pmatrix} 1 \\ -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.71 \\ -0.58 \\ 1.31 \\ 2.89 \end{pmatrix},$$

$$C_c = \hat{M}_{MVP}C = \begin{pmatrix} 0.71 & 0 & 0.71 & 0 \\ 0.58 & 1.16 & -0.58 & 0 \\ 0.71 & -0.71 & -0.71 & -0.11 \\ 0.58 & -0.58 & -0.58 & 1.73 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.16 \\ -0.82 \\ 1.15 \end{pmatrix}.$$

Następnie wykonajmy dzielenie perspektywiczne:

$$A_{NDC} = \begin{pmatrix} -0.71/1.73 \\ -1.75/1.73 \\ -1.52/1.73 \end{pmatrix} \approx \begin{pmatrix} -0.41 \\ -1 \\ -0.06 \end{pmatrix},$$

$$B_{NDC} = \begin{pmatrix} 0.71/2.89 \\ -0.58/2.89 \\ -0.11/2.89 \end{pmatrix} = \begin{pmatrix} 0.25 \\ -0.2 \\ 0.45 \end{pmatrix},$$

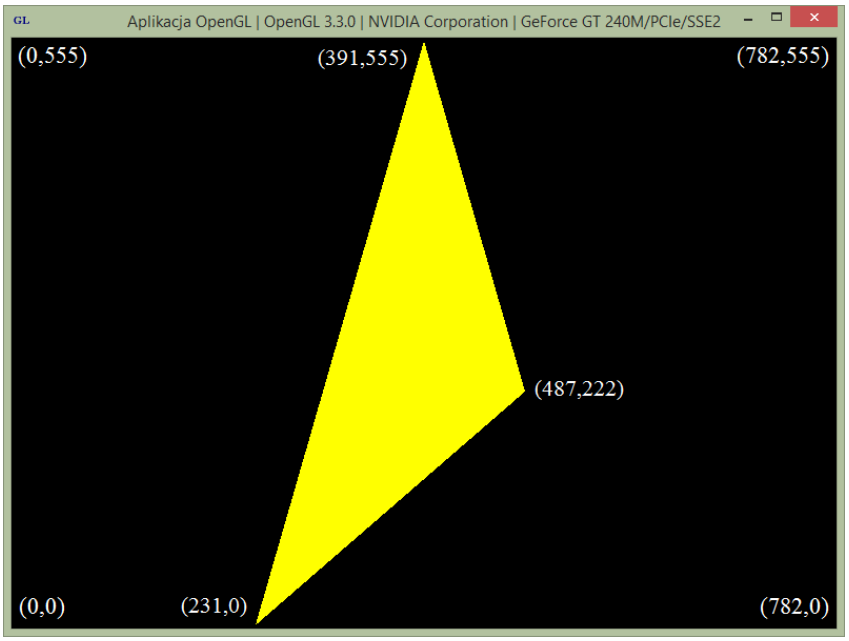
$$C_{NDC} = \begin{pmatrix} 0/1.15 \\ 1.16/1.15 \\ 0.6/1.15 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 1 \\ -0.71 \end{pmatrix}.$$

I przechodzimy do współrzędnych viewportu (wzór (8.32)): WIĘCEJ OPISU I O ZAOKRĄGLENIU I zWIĄZANYMI Z TYM BŁĘDAMI

$$A_v = \begin{pmatrix} 231 \\ 0 \\ 0.47 \end{pmatrix},$$

$$B_v = \begin{pmatrix} 487 \\ 222 \\ 0.73 \end{pmatrix},$$

$$C_v = \begin{pmatrix} 391 \\ 555 \\ 0.14 \end{pmatrix}.$$



Zadania

Pokazać, dla przykładu E = (-1,1,1) i C=(0,0,0), że złożenie obrotu daje tą samą macierz V, co macierz skonstruowana dzięki wersorom.

Skonstruować macierz widoku dla E=(-1,1,1), C=(0,1,0) i U=(0,1,0)

Odbicie względem dowolnej płaszczyzny (odbicie Householdera)

http://en.wikipedia.org/wiki/Transformation_matrix#Examples_in_3D_graphics – to jakaś pochodna operatora gwiazdki!!!!! Dodać po napisaniu o wektorze gwiazdki

Reflection[edit]

Main article: Householder transformation

To reflect a point through a plane ax + by + cz = 0 (which goes through the origin), one can use \mathbf{A} = \mathbf{I}-2\mathbf{NN}^T , where \mathbf{I} is the 3x3 identity matrix and \mathbf{N} is the three-dimensional unit vector for the vector normal of the plane. If the L2 norm of a, b, and c is unity, the transformation matrix can be expressed as:

$$\mathbf{A} = \begin{bmatrix} 1 - 2a^2 & -2ab & -2ac \\ -2ab & 1 - 2b^2 & -2bc \\ -2ac & -2bc & 1 - 2c^2 \end{bmatrix}$$

\mathbf{A} = \begin{bmatrix} 1 - 2 a^2 & - 2 a b & - 2 a c \\ - 2 a b & 1 - 2 b^2 & - 2 b c \\ - 2 a c & - 2 b c & 1 - 2c^2 \end{bmatrix}

Note that these are particular cases of a Householder reflection in two and three dimensions. A reflection about a line or plane that does not go through the origin is not a linear transformation; it is an affine transformation.

2. Wyprowadź macierz przekształcenia współrzędnych NDC do współrzędnych viewportu