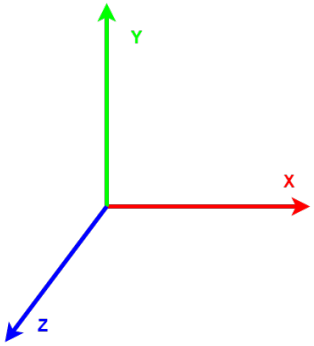
Grafika Komputerowa I

Projekt 4

Temat: Grafika Komputerowa 3D

Wskazówki

Zakłada się prawoskrętny układ współrzędnych sceny, gdzie oś OX skierowana jest w prawo, oś OY w górę, natomiast oś OZ na zewnątrz ekranu w kierunku użytkownika, tak jak jest to widoczne na rysunku 1.



Rysunek 1: Prawoskrętny układ współrzędnych

W aplikacji przydatna będzie reprezentacja pozycji wierzchołków figur na scenie w postaci współrzędnych afinicznych tzn. $[X, Y, Z, W]^T$ gdzie $W = 1$. W przypadku gdy $W = 0$, to zamiast pozycji wektor reprezentuje kierunek. Dzięki temu, wszystkie niezbędne przekształcenia sceny mogą być zapisane w postaci macierzy 4x4.

Poniżej podane są macierze podstawowych transformacji:

$$M_S = \begin{bmatrix} S_X & 0 & 0 & 0 \\ 0 & S_Y & 0 & 0 \\ 0 & 0 & S_Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Macierz skalowania o wartości $[S_X, S_Y, S_Z]^T$ wzdłuż kolejnych osi układu

$$M_T = \begin{bmatrix} 1 & 0 & 0 & T_X \\ 0 & 1 & 0 & T_Y \\ 0 & 0 & 1 & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Macierz translacji o wektor $[T_X, T_Y, T_Z]^T$

$$M_{R_X} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos(\alpha) & -Sin(\alpha) & 0 \\ 0 & Sin(\alpha) & Cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{R_Y} = \begin{bmatrix} Cos(\alpha) & 0 & -Sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ Sin(\alpha) & 0 & Cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{R_Z} = \begin{bmatrix} Cos(\alpha) & -Sin(\alpha) & 0 & 0 \\ Sin(\alpha) & Cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Macierze obrotu kolejno wokół osi OX, OY, OZ o kąt α zgodnie z obrotem wskazówek zegara

$$Proj_{[-1,1]} = \begin{bmatrix} \frac{ctg(\frac{fov}{2})}{aspect} & 0 & 0 & 0 \\ 0 & ctg(\frac{fov}{2}) & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{-2f*n}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Macierz projekcji mapująca współrzędną z po rzutowaniu na wartości z przedziału $[-1, 1]$. n - odległość bliższej ściany obcinania, f - odległość dalszej ściany obcinania, fov - pole widzenia kamery w radianach, *aspect* - proporcja wymiarów ekranu, szerokość do wysokości ($0 < n < f$, $fov \in [0, \pi]$)

Kamera powinna posiadać pozycję P, punkt który obserwuje T oraz wektor do góry U_{world} (należy przyjąć $U_{world} = [0, 1, 0]^T$. Dodatkowo dla każdej kamery definiowane są 3 wektory:

- Kierunek w stronę kamery D - z obserwowanej pozycji T do pozycji kamery P.
- Kierunek w prawo R - iloczyn wektorowy U_{world} i D.
- Kierunek do góry U - iloczyn wektorowy D i R

Wektory te powinny zostać znormalizowane. Wtedy macierz widoku określa się wzorem:

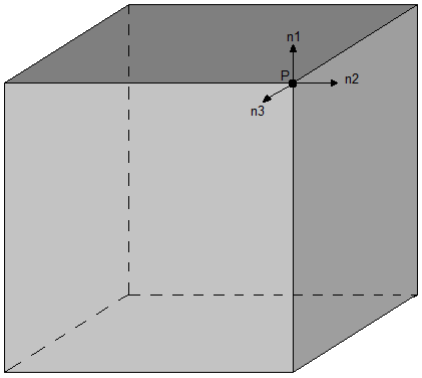
$$View = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ogólnie macierz widoku jest odwrotnością macierzy przekształcenia obiektu kamery.

1 Tworzenie i edycja obiektów

Jak wspomniane zostało wcześniej, obiekty będą reprezentowane jako siatki trójkątów. Dla każdego wierzchołka w takiej siatce potrzebna jest pozycja, wektor normalny, wektor styczny, wektor binormalny oraz współrzędne tekstury niezbędne do oteksturowania danego obiektu. Co więcej, dla każdego trójkąta należy zachować dokładnie tą samą skrętność wierzchołków (winding) np. zgodnie ze wskazówkami zegara. To pomoże przy dalszych operacjach, takich jak obcinanie ścian tylnych. Operacja ograniczy się do sprawdzenia skrętności, jeżeli przykładowo zadbaliśmy, że wierzchołki w każdym trójkącie są uporządkowane zgodnie z ruchem wskazówek zegara (clockwise) a po przekształceniach okaże się, że uporządkowanie będzie odwrotne (anti-clockwise) - taką ścianę należy pomijać przy rysowaniu. Jest to algorytm usuwania ścian tylnych (backface-culling).

Należy zwrócić uwagę, że w miejscach, gdzie bryła nie jest gładka (powierzchnia w danym miejscu jest klasy C_0, czyli jest tam ostra krawędź) wektor normalny nie może być jednoznacznie wyznaczony. Należy wtedy duplikować wierzchołki w taki sposób, aby każda z możliwych normalnych była przyporządkowana dokładnie jednemu wierzchołkowi. Weźmy jako przykład kostkę (tutaj jako ściany użyte są czworokąty, jednak w projekcie powinny to być trójkąty):



Kostka z zaznaczonymi normalnymi n_1, n_2, n_3 dla punktu P

Widać, że punkt P należy rozbić na trzy punkty P_1, P_2 i P_3 z normalnymi odpowiednio n_1, n_2, n_3. Dzięki takiej operacji cieniowanie zostanie wykonane poprawnie (podczas użycia jednej normalnej widoczne byłoby płynne przejście przez róg i krawędź kostki, co jest oczywistym błędem).

Oprócz wektora normalnego wierzchołek powinien również zawierać wektor styczny i binormalny. Dla kostki wektory te wystarczy wpisać ręcznie. Dla zaznaczonej ściany w punkcie P wektorem normalnym będzie n_1 wektorem stycznym będzie n_2 a binormalnym n_3. Kolejność wektora stycznego i binormalnego nie ma większego znaczenia. Ważne jest tylko, żeby każdy z 3 wektorów był prostopadły względem każdego pozostałego. Kolejność wpływa tylko na kierunek wektorów odczytanych z tekstury wektorów normalnych przy mapowaniu normalnych.

Kolejną istotną rzeczą jest tworzenie brył na podstawie ich parametryzacji. Weźmy jako przykład sferę. Jest to dwuwymiarowy obiekt, który będziemy umieszczać na trójwymiarowej scenie. Parametryzacja sfery (nie jedyna, lecz na potrzeby projektu najwygodniejsza) przedstawia się następująco:

$$\begin{cases} x = Rsin\psi cos\phi \\ y = Rcos\psi \\ z = Rsin\psi sin\phi \end{cases}$$

gdzie,

R - promień sfery

ψ - pierwszy parametr z zakresu $[0, \pi]$, odpowiednik szerokości geograficznej (dla 0 będzie biegun północny, natomiast dla π biegun południowy)

ϕ - drugi parametr z zakresu $[0, 2\pi]$, odpowiednik długości geograficznej

Mając daną taką parametryzację, należy próbkować parametry na ich zakresach z krokami d_1, d_2 i wstawiając je do wzoru otrzymamy kolejne punkty na sferze. Ponadto, od d_1 i d_2 zależy gęstość próbkowania, a co za tym idzie, liczba podziałów siatki, która powinna być konfigurowalna w aplikacji. Pozostaje tylko zastanowić się jak połączyć dane punkty aby otrzymać spójną siatkę trójkątów, którą można wyświetlić na scenie.

W przypadku parametryzacji wektory styczny i binormalny można wyznaczyć przez pochodne cząstkowe względem kolejno pierwszego i drugiego parametru. Wektorem normalnym

będzie wtedy iloczyn wektorowy tych wektorów (w odpowiedniej kolejności, tak żeby były skierowane na zewnątrz figury). Dla sfery wektorem normalnym jest po prostu wektor od środka sfery do wierzchołka na jej powierzchni. Bez względu na sposób uzyskania wektorów należy zadbać o ich normalizację.

2 Kolejne kroki potoku renderowania

2.1 Operacje na wierzchołkach

Dla każdego wierzchołka w postaci wektora $[x_m, y_m, z_m, 1]^T$ w przestrzeni modelu wykonywane są przekształcenia przez macierze Modelu, Widoku i Projekcji, przechodząc kolejno z przestrzeni modelu (Model Space) do przestrzeni świata (World Space), przestrzeni widoku (View Space), oraz ostatecznie do 4-wymiarowej jednorodnej przestrzeni obcinania (Homogeneous Clipping Space).

$$Model * [x_m, y_m, z_m, 1]^T = [x_w, y_w, z_w, 1]^T$$
$$View * [x_w, y_w, z_w, 1]^T = [x_v, y_v, z_v, 1]^T$$
$$Proj * [x_v, y_v, z_v, 1]^T = [x_c, y_c, z_c, w_c]^T$$

gdzie

$Model$ - macierz modelu

$View$ - macierz widoku

$Proj$ - macierz projekcji (perspektywy)

x_m, y_m, z_m - współrzędne w przestrzeni modelu

x_w, y_w, z_w - współrzędne w przestrzeni świata

x_v, y_v, z_v - współrzędne w przestrzeni widoku

x_c, y_c, z_c, w_c - 4-wymiarowe współrzędne w jednorodnej przestrzeni obcinania

Oczywiście wszystkie powyższe przekształcenia można wykonać w jednym kroku:

$$Proj * View * Model * [x_m, y_m, z_m, 1]^T = [x_c, y_c, z_c, w_c]^T$$

lub

$$PVM = Proj * View * Model$$
$$PVM * [x_m, y_m, z_m, 1]^T = [x_c, y_c, z_c, w_c]^T$$

Dla niejednorodnej skali (transformacji skalowania o różne wartości wzdłuż różnych osi układu współrzędnych) wektory normalne, styczne i binormalne muszą być przekształcane przez odwrotność transpozycji macierzy transformacji:

$$\left(Model^T\right)^{-1} * [x_m, y_m, z_m, 0]^T = [x_c, y_c, z_c, 0]^T$$

Dotyczy to jednak tylko trzech wymienionych wyżej wektorów. Wszystkie pozostałe wektory (w tym wektory opisujące kierunki) są skalowane przez normalne macierze transformacji (przez macierz widoku i projekcji należy mnożyć tylko pozycję).

Dla podanej macierzy projekcji $Proj_{[-1,1]}$ po przekształceniu do przestrzeni obcinania wartość współrzędnej w_c będzie wynosić dokładnie z_v.

2.2 Operacje na trójkątach

2.2.1 Obcinanie

W przestrzeni obcinania wykonywane jest obcinanie trójkątów wychodzących poza obszar widoczny przez kamerę. Każda z krawędzi trójkąta obcinana jest do 6 hiperpłaszczyzn obcinania w 4-wymiarowej przestrzeni. Każda z hiperpłaszczyzn odpowiada jednej ścianie ostrosłupa pola widzenia kamery. Przestrzeń widoczną przez kamerę można opisać poniższymi nierównościami:

$$\begin{cases} x \geqslant -w \\ x \leqslant w \\ y \geqslant -w \\ y \leqslant w \\ z \geqslant -w \\ z \leqslant w \end{cases}$$

Nierówności te wynikają z tego, że po podziale współrzędnej przez składową w (rzutowanie do przestrzeni trójwymiarowej) chcemy uzyskać wartości z przedziału $[-1, 1]$.

Do obcięcia trójkąta można wykorzystać algorytm obcinania Sutherlanda–Hodgmana. Aby wyznaczyć punkt przecięcia C odcinka AB z hiperpłaszczyzną można wyznaczyć długość rzutu ze znakiem wektorów PA oraz PB (gdzie P to punkt na płaszczyźnie najbliższy środkowi układu współrzędnych) na wektor normalny hiperpłaszczyzny i liniowo zinterpolować się pomiędzy wynikami.

Czterowymiarową powierzchnię można zapisać równaniem implicit w postaci:

$$S([x, y, z, w]^T) = ax + by + cz + dw - e = 0$$

gdzie

$[a, b, c, d]^T$ - wektor normalny do płaszczyzny

e - odległość płaszczyzny od środka układu współrzędnych przemnożona przez długość wektora normalnego

Wstawiając w równanie $S(V)$ dowolne punkty V otrzymamy ich odległość ze znakiem od płaszczyzny przemnożoną przez długość wektora normalnego. Jest to dokładnie rzut wektora V na wektor normalny płaszczyzny. W przypadku interpolacji pomiędzy dwoma odległościami od tej samej płaszczyzny, przeskalowanie o stałą (długość wektora normalnego) nie zmieni wyniku, więc można ten fakt zignorować. W przypadku nierówności należy zwrócić uwagę na zwrot wektora normalnego i ewentualnie zamienić znaki w równaniu na przeciwne, tak aby punkty spełniające nierówność miały dodatni znak odległości ze znakiem. **Punkty V dla których odległość będzie miała ujemny znak leżą za płaszczyzną (znajdują się poza przestrzenią widoczną przez kamerę), natomiast pozostałe punkty znajdują się przed płaszczyzną (znajdują się wewnątrz przestrzeni widocznej przez kamerę).**

W przypadku powyższych płaszczyzn otrzymujemy następujące wzory na odległości ze znakiem:

$$dist_i([x, y, z, w]^T) = \begin{cases} w - x, & i = 1 \\ w + x, & i = 2 \\ w - y, & i = 3 \\ w + y, & i = 4 \\ w - z, & i = 5 \\ w + z, & i = 6 \end{cases}$$

Mając powyższe wzory możemy w prosty sposób wyznaczyć punkt przecięcia C odcinka AB z i-tą płaszczyzną obcinania poprzez zwykłą interpolację liniową:

$$d_A = dist_i(A)$$

$$d_B = dist_i(B)$$
$$d_C = \frac{d_A}{d_A - d_B}$$
$$C = A * (1 - d_C) + B * (d_C)$$

Punkt przecięcia należy liczyć tylko w przypadku gdy odległości mają przeciwne znaki (leżą po przeciwnych stronach płaszczyzny obcinania).

W taki sam sposób można wyznaczyć dowolny inny atrybut wierzchołka, np. kolor podstawiając w ostatnim równaniu pod A kolor wierzchołka A, a pod B kolor wierzchołka B.

Z powyższą funkcją $dist_i$ bardzo łatwo można zaimplementować algorytm obcinania Sutherlanda–Hodgmana. Dozwolona jest jednak implementacja dowolnego algorytmu obcinania wielokątów.

Po obcięciu trójkąta może on zostać całkowicie odrzucony w przypadku gdy leży poza polem widzenia. W pozostałych przypadkach powstały wielokąt może mieć od 3 do 9 wierzchołków. W przypadku implementacji wypełniania tylko dla trójkątów należy podzielić go na trójkąty. Powstały wielokąt będzie wielokątem wypukłym, więc podział jest trywialny. Najprostszym (chociaż nie najlepszym) sposobem jest wybranie jednego wierzchołka i poprowadzenie z niego przekątnych do wszystkich pozostałych wierzchołków.

2.2.2 Rzutowanie do przestrzeni ekranu

Po obcięciu wszystkich trójkątów (możliwe jest wygenerowanie dodatkowych trójkątów lub całkowite odrzucenie innych) wykonywane jest rzutowanie z 4-wymiarowej jednorodnej przestrzeni obcinania do przestrzeni znormalizowanych współrzędnych urządzenia (Normalized Device Coordinates - NDC). Każda ze współrzędnych przyjmuje w niej wartość z zakresu od -1 do 1.

$$\left[\frac{x_c}{w_c}, \frac{y_c}{w_c}, \frac{z_c}{w_c}, \frac{w_c}{w_c}\right]^T = [x_{NDC}, y_{NDC}, z_{NDC}, 1]^T$$

Następnym krokiem jest przejście do przestrzeni ekranu. Współrzędna x zostaje zamieniona na wartość z przedziału $[0, W]$, y do $[0, H]$, gdzie W i H to odpowiednio szerokość i wysokość ekranu w pikselach. natomiast z do przedziału $[0, 1]$.

$$[x_s, y_s, z_s]^T = \left[\frac{(x_{NDC} + 1)W}{2}, \frac{(-y_{NDC} - 1)H}{2}, \frac{z_{NDC} + 1}{2}\right]^T$$

Warto zauważyć, że przestrzeń ekranu jest przestrzenią trójwymiarową. Współrzędne x_s oraz y_s wskazują konkretny piksel na ekranie, natomiast współrzędna z_s opisuje głębię piksela. Będzie ona niezbędna do algorytmu buforowania składowej głębokości, który zapewni odpowiednią kolejność wyświetlania obiektów na scenie.

Jeśli włączone jest obcinanie ściany tylnej, to ostatnim krokiem jest sprawdzenie czy dany trójkąt ułożony jest tyłem do ekranu. Jeśli tak, to należy go odrzucić i nie przetwarzać go dalej. Można w tym celu wykorzystać iloczyn wektorowy.

2.3 Operacje na pikselach

2.3.1 Rasteryzacja

Rasteryzację należy przeprowadzić algorytmem skan-linii. Można zoptymalizować algorytm pod trójkąty. Dla każdego piksela trójkąta niezbędne będzie wyznaczenie jego składowej głębi oraz pozostałych atrybutów takich jak kolor czy wektor normalny. W tym celu niezbędna będzie interpolacja.

Można zastosować 3 razy interpolację liniową (raz pomiędzy dwoma naprzeciwległymi krawędziami trójkąta, a następnie drugi raz dla każdego piksela pomiędzy krawędziami), lub interpolację barycentryczną. Wybór jest dowolny, chociaż w przypadku skan-linii pierwsze rozwiązanie bardzo dobrze się wpasowuje w algorytm.

Interpolacja liniowa składowej głębokości:

$$z_{s12} = z_{s1} * (1 - q) + z_{s2} * q$$

gdzie,

z_{s12} - zinterpolowana składowa głębokości pomiędzy z_{s1} i z_{s2}

$q \in [0, 1]$

Interpolacja barycentryczna składowej głębokości:

$$z_{s123} = z_{s1} * b_1 + z_{s2} * b_2 + z_{s3} * b_3$$

gdzie,

z_{s123} - zinterpolowana składowa głębokości

z_{si} - składowa głębokości we współrzędnych ekranu dla i-tego wierzchołka

b_i - współrzędna barycentryczna trójkąta w przestrzeni ekranu ($b_1 + b_2 + b_3 = 1$)

Współrzędne barycentryczne łatwo jest wyznaczyć ze stosunku pól poszczególnych trójkątów. Weźmy trójkąt z punktami $v_{s1} = (x_{s1}, y_{s1}, z_{s1})$, $v_{s2} = (x_{s2}, y_{s2}, z_{s2})$, $v_{s3} = (x_{s3}, y_{s3}, z_{s3})$, oraz punkt $v_s = (x_s, y_s, z_s)$. Znamy tylko jego współrzędne x_s oraz y_s (są to współrzędne kolejnych pikseli leżących wewnątrz dwuwymiarowego trójkąta określonego punktami (x_{si}, y_{si}).

$$b_i = \frac{a_i}{a}$$

gdzie,

a_1 - pole trójkąta v_s, v_{s2}, v_{s3}

a_2 - pole trójkąta v_s, v_{s3}, v_{s1}

a_3 - pole trójkąta v_s, v_{s1}, v_{s2}

Wyznaczona składowa głębokości powinna zostać użyta w algorytmie buforowania składowej głębi (z-buffering).

Jeśli w powyższych wzorach zamiast składowej głębi będziemy interpolować dowolny inny atrybut wierzchołków trójkąta, to wynik okaże się niepoprawny w przypadku gdy wierzchołki znajdują się w różnej odległości do ekranu. Jest to interpolacja bez korekty perspektywy. Aby uzyskać poprawny wynik, należy w interpolacji uwzględnić głębokość poszczególnych pikseli (dokładnie odwrotność współrzędnej z w przestrzeni widoku). Wartość odwrotna do potrzebnej zapisana jest we współrzędnej w przestrzeni obcinania.

Interpolacja liniowa dowolnego atrybutu wierzchołka:

$$atr_{12} = \frac{atr_1 * \frac{1-q}{w_{c1}} + atr_2 * \frac{q}{w_{c2}}}{\frac{1-q}{w_{c1}} + \frac{q}{w_{c2}}}$$

Interpolacja barycentryczna dowolnego atrybutu wierzchołka:

$$atr_{123} = \frac{atr_1 * \frac{b_1}{w_{c1}} + atr_2 * \frac{b_2}{w_{c2}} + atr_3 * \frac{b_3}{w_{c3}}}{\frac{b_1}{w_{c1}} + \frac{b_2}{w_{c2}} + \frac{b_2}{w_{c2}}}$$

Przy interpolacji wektorów normalnych należy w każdym kroku przeprowadzić jego normalizację.

2.3.2 Cieniowanie

Należy zwrócić uwagę na normalizację wektorów. Wszystkie wektory z daszkiem (np. $\hat{N}$) są znormalizowane. Kolory mają wartości z przedziału $[0-1]$**.**

Każdy materiał posiada 4 zdefiniowane stałe wpływające na oświetlenie.

k_s - współczynnik (kolor) składowej odbić lustrzanych (specular)

k_d - współczynnik (kolor) składowej rozproszonej (diffuse)

k_a - współczynnik (kolor) składowej oświetlenia otoczenia (ambient)

α - współczynnik połysku (shininess)

$$C = k_a * I_a + \sum_{I \in \text{Lights}} \left(k_d \left(\hat{L_I} \cdot \hat{N} \right) I_d + k_s \left(\hat{R} \cdot \hat{V} \right)^{\alpha} I_s \right) * I_f$$

gdzie,

C - finalny kolor piksela

Lights - zbiór świateł

I_a - kolor światła otoczenia (ambient light)

I_d - kolor światła rozproszonego (diffuse light)

I_s - kolor światła odbić lustrzanych (specular)

$\hat{L_I}$ - wersor od piksela (we współrzędnych świata) do światła I

$\hat{N}$ - wersor normalny dla danego piksela

$\hat{V}$ - wersor do obserwatora

$\hat{R}$ - wersor odbicia wersora do światła względem wersora normalnego

$$\hat{R} = 2 \left(\hat{L_I} \cdot \hat{N} \right) \hat{N} - \hat{L_I}$$

I_f - Współczynnik osłabienia światła wraz z odległością (atenuacja)

$$I_f = \frac{1}{A_C + A_L * Dist + A_Q * Dist^2}$$

gdzie

A_C, A_L, A_Q - stałe (przykładowe wartości: $A_C = 1$, $A_L = 0.09$, $A_Q = 0.032$

$Dist$ - odległość od źródła światła

2.3.3 Teksturowanie

Każdy wierzchołek powinien mieć atrybut będący współrzędną tekstury. Są to dwie wartości które informują który piksel z tekstury powinien zostać wyświetlony dla danego piksela. Zazwyczaj są to wartości z przedziału $[0, 1]$ gdzie para $(0, 0)$ oznacza lewy górny róg tekstury, natomiast $(1, 1)$ prawy górny. Dla każdego piksela wyświetlonego na ekranie w wyniku rasteryzacji należy oprócz wektora normalnego w taki sam sposób zinterpolować współrzędne tekstury oraz pobrać odpowiednią wartość z obrazka. Kolor ten należy podstawić pod zmienną k_d w cieniowaniu.

2.3.4 Mapowanie normalnych

Do algorytmu mapowania normalnych tak jak przy teksturowaniu potrzebne są współrzędne tekstury, jak również wektor normalny, wektor styczny oraz wektor binormalny. Wszystkie są interpolowane w ten sam sposób podczas rasteryzacji.

$$\hat{N}_M = normalize\left([M_R, M_G, M_B]^T\right)$$

$$TBN = \begin{bmatrix} \hat{T}_x & \hat{B}_x & \hat{N}_x \\ \hat{T}_y & \hat{B}_y & \hat{N}_y \\ \hat{T}_z & \hat{B}_z & \hat{N}_z \end{bmatrix}$$

$$\hat{N}' = normalize\left(TBN * \hat{N}_M\right)$$

gdzie,

M_R, M_G, M_B - wartości z mapy normalnych przeskalowane do przedziału $[-1, 1]$

$\hat{N}_M$ - wersor normalny odczytany z mapy normalnych

$\hat{N}'$ - nowy wersor normalny w pikselu uzyskany z mapowania normalnych

$\hat{N}$ - wersor normalny w pikselu

$\hat{T}$ - wersor styczny w pikselu

$\hat{B}$ - wersor binormalny w pikselu

$$M_R = \frac{Tex_R}{255} * 2 - 1$$

$$M_G = \frac{Tex_B}{255} * 2 - 1$$

$$M_B = \frac{Tex_B}{255} * 2 - 1$$

gdzie

Tex_R, Tex_G, Tex_B - wartości koloru mapy normalnych z przedziału $[0, 255]$ dla piksela uzyskana z tekstury przy pomocy współrzędnych tekstury

3 Przydatne linki

- `https://learnopengl.com/`
- `http://fabiensanglard.net/polygon_codec/clippingdocument/p245-blinn.pdf`