Karol Sobiesiak

Politechnika Gdańska, Wydział ETI, Inteligentne Systemy Interaktywne

TECHNIKI SPRZĘTOWEJ TESELACJI DO WYGŁADZANIA I USZCZEGÓŁAWIANIA MODELI TRÓJWYMIAROWYCH

Streszczenie

DirectX 11 oraz OpenGL 4 wprowadziły do potoku renderującego bardzo oczekiwaną teselację. Wraz z upływem czasu coraz więcej silników graficznych zostaje dostosowywanych do wspierania tej technologii, więc nie można pozostawać wobec niej obojętnym. W poniższym artykule przedstawiona zostanie koncepcja teselacji od strony teoretycznej oraz praktycznej. Opisany zostanie nowy potok renderujący, a w szczególności dedykowane teselacji programy cieniujące. Główną część artykułu stanowić będzie opis różnych technik, za pomocą których uzyskiwane są efekty takie jak wygładzanie powierzchni czy odwzorowywanie przemieszczeń (ang. *Displacement Mapping*), które pozwalają na efektowne uszczegóławianie modeli trójwymiarowych.

1. WSTĘP

Od początków powstania gier trójwymiarowych, modele w nich występujące cierpiały na bardzo poważną chorobę przejawiającą się „kanciastością". Nawet na tą chwilę w najnowszych produkcjach ten niepożądany artefakt jest widoczny na wielu modelach. Winną za taki stan rzeczy jest sama koncepcja grafiki trójwymiarowej opartej na siatce wielokątów. Konsekwencje takiego wyboru widać niemal we wszystkich grach. Każda opisana w ten sposób sfera będzie jedynie wielościanem aproksymującym jej kształt. Teoretycznie wszystkie kanciastości można by całkowicie wyeliminować, gdyby gęstość siatki modelu pokrywała się z gęstością pikseli na ekranie wyświetlacza. Jednak koszt powiązany z przetwarzaniem takiego modelu w potoku renderującym, wyposażonym jedynie w shadery wierzchołków i geometrii, byłby ogromny. Problem na szczęście został dostrzeżony przez producentów kart graficznych i twórców API. Lekarstwem na wszelkie niechciane kanciastości okazała się Teselacja. Jej sprzętowa implementacja pozwoliła na efektywne zagęszczanie siatki modeli trójwymiarowych bez radykalnych spadków wydajności.

Sama teselacja, powodująca jedynie zagęszczanie geometrii, nie jest jednak bardzo pomocna. Konieczna jest kontrola nad przebiegiem takiego procesu. DirectX w wersji 11 oraz OpenGL w wersji 4.0 oficjalnie wprowadziły nowe shadery, które pozwalają programiście parametryzować pracę teselatora oraz manipulować jego danymi

wynikowymi. To otworzyło drogę dla całej masy algorytmów, z których wiele powstało na długo przed rozważaniem sprzętowej teselacji, a które implementuje się z wielką łatwością.

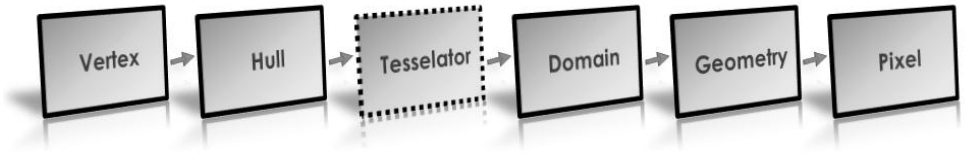
W niniejszej pracy opisane zostaną metody pozwalające wykorzystać potencjał teselacji do zagęszczania i wygładzania siatki wielokątów. Dodatkowo zaprezentowana zostanie technika odwzorowywania przemieszczeń, która w połączeniu z metodami wygładzania oraz odwzorowywania wypukłości zwraca ciekawe rezultaty. Na początek jednak przybliżony zostanie nowy potok renderujący, który został dostosowany do nowej technologii.

2. TESELACJA I JEJ MIEJSCE W POTOKU RENDERUJĄCYM

Teselacja w grafice trójwymiarowej jest algorytmem podziału wielokątów na mniejsze, z zachowaniem wewnętrznej topologii. Podział ten jest zwykle dokonywany na trójkątach. W przypadku, gdy wielokątem źródłowym jest czworokąt przy podziale następuje automatyczna triangulacja, która jednak nie przeszkadza w operowaniu na czworokątach.

Specjalnie na potrzeby teselacji został zdefiniowany nowy kształt podstawowy, który nie jest utożsamiany z żadną figurą geometryczną, a jest jedynie informacją o ilości wierzchołków i ich atrybutach, które zostaną przesłane przez potok. Został on nazwany *Patch*, co można utożsamić z płatem bądź łatą (w dalszej części artykułu używana będzie nazwa płat). W przypadku chęci skorzystania z teselacji, jest to jedyny dostępny sposób określania kształtu geometrycznego. Liczbę wierzchołków w płacie można deklarować ręcznie, jednak istnieje górna granica określona przez API. W związku z możliwością ręcznego określania rozmiaru płatu nasuwa się pytanie – Co w przypadku, gdy przesyłamy więcej niż cztery wierzchołki na płat? Czy procesor graficzny połączy je w pewien sposób i zwróci bliżej nieokreślony wielokąt? Otóż nie. Karta graficzna potrafi przetwarzać jedynie wielokąty w postaci trójkątów oraz czworokątów (które w ostateczności i tak dzielone są na trójkąty). Jaki jest zatem cel przesyłania większej ilości wierzchołków na płat? Otóż nie wszystkie wierzchołki muszą być częścią jakiegoś wielokąta. Można np. przesłać 16 wierzchołków na płat i cztery z nich wykorzystać do narysowania czworokąta, a 12 pozostałych potraktować jako punkty kontrolne do obliczeń powierzchni Béziera. Daje to oczywiście szerokie pole do popisu dla algorytmów operujących na siatce geometrii, do których zaliczyć można m.in. *Catmull-Clark subdivision surface*, powierzchnie NURBS czy płaty Béziera. Nie wszystkie jednak dobrze współpracują z aktualną architekturą potoku, dlatego też powstają zmodyfikowane wersje dostosowane do stawianych wymagań jak np. [1]. Niekiedy też nie da się dostosować algorytmu do dotychczasowych danych i trzeba modyfikować np. bufor indeksów jak w PN-AEN Triangles [2]. Nie wszystkie też nadają się do zastosowań w grach komputerowych z racji ich złożoności obliczeniowej, będącej podstawowym kryterium wyboru w aplikacjach generujących obrazy w czasie rzeczywistym.

Wprowadzenie sprzętowej teselacji wiązało się z pewnymi zmianami w samym potoku renderującym. Dotychczasowe shadery pozostały na swoim miejscu wraz z całą ich funkcjonalnością i przeznaczeniem. Najistotniejszą zmianą jest wprowadzenie trzech nowych etapów ściśle powiązanych z teselacją. Wpasowują się one pomiędzy etap przetwarzania wierzchołków, a konsolidacji geometrii co uwidacznia rysunek 1.



Rys.1. Potok renderujący (uproszczony), występujący w Direct3D 11 i OpenGL 4

Wszystkie powyższe jednostki, poza teselatorem, są programowalne. Shadery wierzchołków, geometrii i pikseli nie dotyczą bezpośrednio teselacji i nie zmieniły się zanadto, więc nie będą tutaj szczegółowo omawiane.

2.1. Shader kontroli teselacji

Program kontroli teselacji w Direct3D został nieco enigmatycznie nazwany *Hull Shader*. W OpenGL natomiast określany jest mianem *Tessellation Control Shader*. W ogólności program ten zajmuje się poniższymi zadaniami:

- Przetwarza wierzchołki zawarte w płacie
- Kontroluje stopień teselacji wewnętrznej i zewnętrznej[1]
- Określa sposób rozstawienia nowych wierzchołków (Direct3D)[2]

Etap ten następuje bezpośrednio po shaderze wierzchołków i jest niejako jego rozwinięciem. Podczas gdy shader wierzchołków przetwarza wszystkie wierzchołki niezależnie i bez współdzielenia danych, shader kontroli teselacji posiada informację o wszystkich wierzchołkach wchodzących w skład płata, co czyni go nieco bardziej podobnym do shadera geometrii.

Shader kontroli teselacji składa się niejako z dwóch etapów. Pierwszy polega na wykonaniu tego samego programu dla każdego z wierzchołków w płacie z osobna. Dzięki dostarczonemu ID można określić, który wierzchołek jest aktualnie przetwarzany. Dodatkowo, istnieje możliwość przesłania dalej do potoku mniejszej liczby wierzchołków niż jest rzeczywiście w płacie, co ogranicza równocześnie liczbę wywołań shadera. Drugi etap natomiast dotyczy obliczeń samego płata. Jest więc wykonywany raz na płat i następuje bezpośrednio po przetworzeniu wierzchołków wchodzących w jego skład. Twórcy OpenGL i Direct3D niestety nie byli zgodni co do rozwiązań i etapy te w obu przypadkach prezentują się inaczej.

W Direct3D wybrano nieco bardziej eleganckie rozwiązanie rozdzielając oba etapy na dwie funkcje. Przetwarzanie wierzchołków odbywa się w głównej funkcji, natomiast przed jej definicją należy umieścić deklarację funkcji wykonującej obliczenia dla każdego płata (zwanej *Patch Constant Function*), po czym ją zdefiniować. Oczywiście przetworzone wierzchołki można przesłać do funkcji operującej na płatach poprzez parametr.

W OpenGLu z kolei wybrano drogę pozwalającą programiście samemu określić, w którym momencie chce mieć dostęp do przetworzonych wierzchołków pierwszego etapu. Nie ma tu wyraźnej granicy jak w Direc3D, gdyż dostępna jest jedna funkcja wejścia. Programista w momencie, gdy chce mieć pewność, że wszystkie wierzchołki zostały przetworzone i chce uniknąć hazardu podczas operowania na nich, powinien wywołać wbudowaną funkcję *barrier()*. Zatrzyma ona przetwarzanie danego wierzchołka do

[1,2] Więcej na ten temat w punkcie 2.4.

momentu, gdy reszta będąca częścią płatu nie dojdzie do tego miejsca. Gdy już wszystkie je osiągną, dalsze przetwarzanie jest już operowaniem na płacie, jako że dostępne są już wszystkie przetworzone wierzchołki. Trzeba jednak pamiętać, by w tym momencie ustawić instrukcję warunkową, która zleci tylko jednemu procesowi dalsze przetwarzanie (inaczej wszystkie wykonywały by to samo). Można w tej sytuacji skorzystać z ID wierzchołka.

2.2. Teselator

Drugim etapem następującym bezpośrednio po shaderze kontroli teselacji jest sama teselacja. Etap ten nie jest programowalny, a jedynie konfigurowalny. Głównym jego zadaniem jest wytworzenie dodatkowych wierzchołków, które zostaną obsłużone w dalszych etapach potoku.

2.3. Shader ewaluacji teselacji

Ostatnim etapem powiązanym z teselacją jest shader ewaluacji teselacji. OpenGL w tym przypadku określa go mianem „*Tessellation Evaluation Shader*”, natomiast Direct3D „*Domain Shader*”. Program ten otrzymuje dane z obu poprzednich etapów i operuje na nich. Ponadto odpowiedzialny jest za poniższe czynności:

- Określa rodzaj prymitywu poddawanego teselacji (trójkąt, czworokąt, izolinia) oraz dostarcza wewnętrzne współrzędne – dwu (czworokąt, izolinia) bądź trójskładowe (trójkąt – współrzędne barycentryczne)
- Określa sposób rozstawienia nowych wierzchołków (OpenGL) [3]
- Kontroluje kształt steselowanego prymitywu za pomocą punktów kontrolnych obliczonych w shaderze kontroli teselacji

Shader ewaluacji teselacji otrzymuje od teselatora nowo wygenerowane wierzchołki i wykonuje się niezależnie dla każdego z nich. Lokalną pozycję nowych wierzchołków można określić za pomocą dostarczonych współrzędnych. Pozwalają one również na prostą interpolację atrybutów wierzchołków na całej powierzchni. Ewentualne punkty kontrolne otrzymane od shadera kontroli teselacji służą do wyznaczania właściwego kształtu steselowanej powierzchni. Po wykonanych obliczeniach, wierzchołki można przemnożyć przez odpowiednie macierze w celu przeniesienia ich do współrzędnych ograniczonych (ang. *clip coordinates*).

W bieżącym shaderze implementowane są funkcje odpowiedzialne za przekształcenia powierzchni na dowolną inną reprezentację np. płaty Béziera. W tym miejscu realizuje się również efekt odwzorowania przemieszczeń. Shader ten jest zatem sporym obciążeniem dla potoku, gdyż zwykle wykonuje najwięcej pracy w cyklu teselacji. Należy go zatem implementować z rozwagą i w miarę możliwości optymalizować kod.

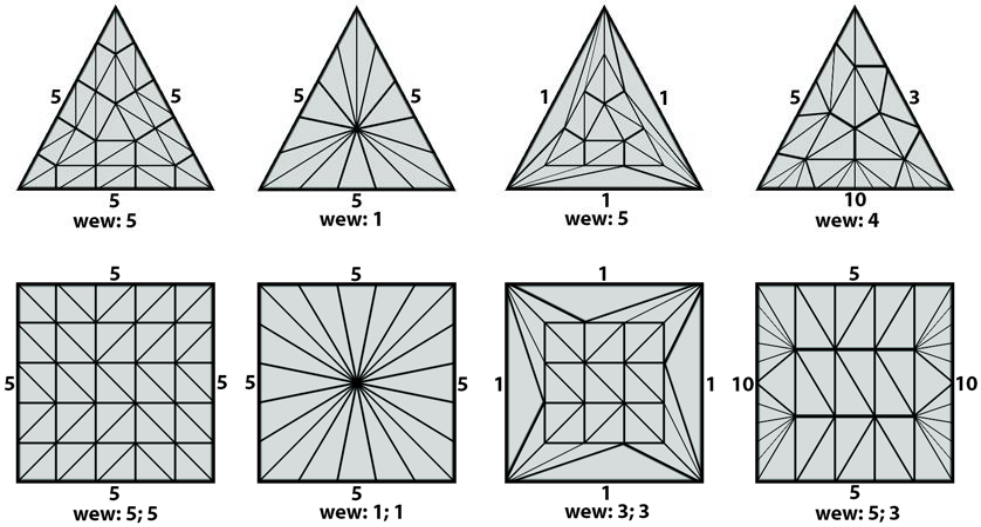
2.4. Właściwości teselacji

Shader kontroli teselacji pozwala kontrolować stopień w jakim dany prymityw zostanie podzielony. Do dyspozycji są dwa rodzaje teselacji – jeden decyduje o stopniu teselacji wewnątrz wielokąta, a drugi na jego krawędziach. Ponadto każdej krawędzi można przypisać inny stopień teselacji. Czworokąty mają również dwa współczynniki teselacji wewnętrznej, które decydują o podziale wzdłuż naprzeciwległych krawędzi. W

[3] Więcej na ten temat w punkcie 2.4.

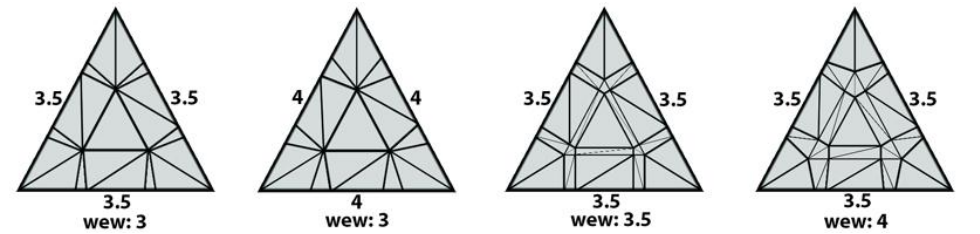
przypadku teselacji czworokątów, ich wewnętrzna struktura zostaje automatycznie podzielona na trójkąty.

Kolejnym ważnym elementem jest deklaracja sposobu podziału wielokąta. Do wyboru jest tryb całkowitoliczbowy lub zmiennoprzecinkowy (w dwóch wariantach – parzysty i nieparzysty). Rysunek 2 pokazuje efekt zastosowania teselacji na trójkącie i czworokącie dla różnych współczynników w trybie całkowitoliczbowym.



Rys.2. Trójkąt i czworokąt dla trybu całkowitoliczbowego.

Tryb całkowitoliczbowy nie pozwala na łagodne przejścia między kolejnymi poziomami teselacji. Istnieje jednak możliwość stosowania liczb zmiennoprzecinkowych. W takim wypadku nowo powstałe trójkąty wyłaniają się z innych krawędzi i powiększają w miarę zwiększania współczynnika. Jest to bardzo przydatna cecha, gdyż geometria może być w ten sposób zagęszczana płynnie. Rysunek 3 pokazuje efekt zastosowania tego trybu.



Rys.3. Teselacja trójkąt dla trybu zmiennoprzecinkowego (nieparzystego).

Można w ten sposób płynnie przechodzić między kolejnymi poziomami LOD (ang. *Level of Detail*) modelu nie martwiąc się o (często stosowane w grach) gwałtowne podmiany modeli różnej jakości.

3. TECHNIKI WYGŁADZANIA

W bieżącym punkcie opisane zostaną trzy techniki wygładzania siatki wielokątów, które z racji relatywnie niewielkiego kosztu obliczeniowego nadają się do zastosowań w grach komputerowych. W przypadku chęci uzyskania bardziej szczegółowych informacji, zalecane jest zapoznanie się ze źródłowymi publikacjami, gdyż ich autorzy przysporzyli sobie wiele trudu opracowując owe techniki.

3.1. PN Triangles

Jedną z najbardziej popularnych technik wykorzystujących możliwości teselacji do wygładzania siatki jest PN Triangles [3] (w rozwinięciu ang.: *Curved Point-Normal Triangles*). Została opracowana jeszcze przed wprowadzeniem sprzętowej teselacji, jednak doskonale się z nią integruje. Zasada jej działania opiera się na trójkątnych płatach Béziera. Największą jej zaletą jest fakt, że nie wymaga żadnej dodatkowej informacji poza pozycjami i wektorami normalnymi wszystkich punktów trójkąta, a bezpośredni dostęp do nich można uzyskać w shaderze kontroli teselacji. Fakt ten oznacza, że nie ma konieczności dokonywania żadnych modyfikacji odnośnie posiadanych danych samego modelu. Wystarczy tylko dostosować kod shaderów, co zwykle nie przysparza żadnych problemów. Jest to niezwykle istotne, gdyż koszt związany z ewentualnymi zmianami, jakie grafik musiał by wprowadzać w modelu, może zadecydować o wykorzystaniu danego efektu. Dzięki tej cesze PN Triangles można zastosować do tysięcy gotowych modeli trójwymiarowych i tym samym „wyleczyć" je z kanciastości.

3.1.1. Obliczenia kubicznego trójkąta Béziera

Trójkątne płaty Béziera są mniej popularną wersją szeroko rozpowszechnionych płatów Béziera. Działają one na tej samej zasadzie tzn. za pomocą punktów kontrolnych definiują kształt powierzchni – w tym wypadku trójkąta. PN Triangles do jego opisu wykorzystuje kubiczny[2] trójkąt Béziera opisany wzorem:

$$B(u,v,w) = \sum_{i+j+k=3} b_{ijk} \frac{3!}{i!\,j!\,k!} u^i v^j w^k \tag{3.1}$$

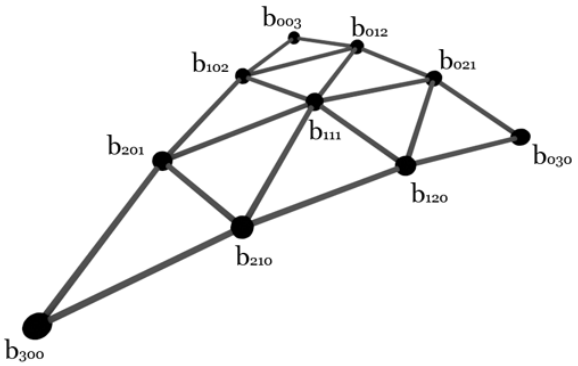
gdzie: u,v,w – współrzędne barycentryczne trójkąta ($u+v+w=1$),
b_{ijk} – punkty kontrolne trójkąta Béziera.

który po rozwinięciu przyjmuje postać:

$$\begin{aligned} B(u,v,w) = {} & b_{300}w^3 + b_{030}u^3 + b_{300}v^3 \\ & + b_{210}3w^2u + b_{120}3wu^2 + b_{201}3w^2v \\ & + b_{021}3u^2v + b_{102}3wv^2 + b_{012}3uv^2 \\ & + b_{111}6wuv \end{aligned} \tag{3.2}$$

[2] Krzywe Béziera są szczególną postacią wielomianu określonego stopnia. Najszerzej wykorzystywanymi są krzywe stopnia trzeciego – kubiczne.

Uzyskana funkcja jest w stanie przekształcić płaską steselowaną siatkę trójkąta do trójkątnego płatu Béziera. Punkty kontrolne rozkładają się na trójkącie jak pokazuje rysunek 4.



Rys.4. Rozkład punktów kontrolnych kubicznego trójkąta Béziera tworzących siatkę kontrolną.

Punkty bazowe trójkąta pokrywają się z punktami b_{300}, b_{030} b_{003} i nie podlegają modyfikacjom. Pozostałe punkty kontrolne decydują o odkształceniach powierzchni. Trzeba jednak pamiętać, że zależy nam, by dwa trójkąty będące w sąsiedztwie były nadal połączone krawędziami, mimo odkształceń. W takim wypadku należy uzależnić punkty kontrolne od wektorów normalnych znajdujących się na trzech bazowych wierzchołkach. Znając współrzędne punktów bazowych trójkąta i odpowiadające im wektory normalne, wewnętrzne punkty kontrolne wyznacza się wg wzoru 3.3. Szczegółowe wyprowadzenie wszystkich punktów, które jest stosunkowo proste, przedstawiono w [3].

$$
\begin{aligned}
&b_{300} = P_1 \qquad b_{030} = P_2 \qquad b_{003} = P_3 \\
&w_{ij} = (P_j - P_i) \cdot N_i \\
&b_{210} = (2P_1 + P_2 - w_{12}N_1)/3 \qquad b_{120} = (2P_2 + P_1 - w_{21}N_2)/3 \\
&b_{021} = (2P_2 + P_3 - w_{23}N_2)/3 \qquad b_{012} = (2P_3 + P_2 - w_{32}N_3)/3 \\
&b_{102} = (2P_3 + P_1 - w_{31}N_3)/3 \qquad b_{201} = (2P_1 + P_3 - w_{13}N_1)/3 \\
&E = (b_{210} + b_{120} + b_{021} + b_{012} + b_{102} + b_{201})/6 \\
&V = (P_1 + P_2 + P_3)/3 \\
&b_{111} = E + (E - V)/2
\end{aligned}
\tag{3.3}
$$

gdzie: P – współrzędne wierzchołka,
N – wektor normalny wierzchołka.

W tym momencie dostępne są wszystkie niezbędne dane potrzebne do obliczeń powierzchni wg wzoru 3.2.

3.1.2. Obliczenia wektorów normalnych

Kolejnym krokiem jest wyznaczenie wektorów normalnych na powierzchni odkształconego trójkąta. W przypadku bardzo prostych modeli można zastosować interpolację liniową, jednak dla bardziej złożonych siatek technika PN Triangles zaleca stosowanie funkcji wielomianowej drugiego stopnia danej wzorem:

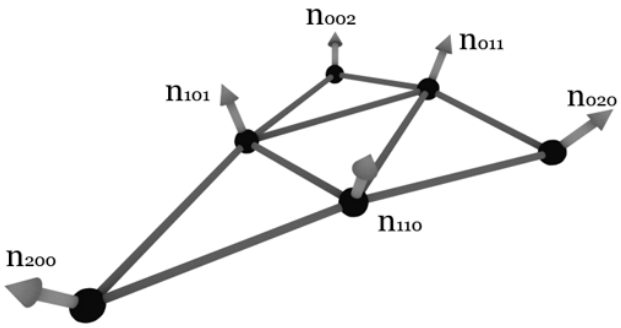
$$N(u,v,w) = \sum_{i+j+k=2} n_{ijk} u^i v^j w^k \tag{3.4}$$

gdzie: u,v,w – współrzędne barycentryczne trójkąta,
n_{ijk} – punkty kontrolne.

która po rozwinięciu przyjmuje postać:

$$\begin{aligned} N(u,v,w) = n_{200}w^2 + n_{020}u^2 + n_{002}v^2 + \\ + n_{110}wu + n_{011}uv + n_{101}wv \end{aligned} \tag{3.5}$$

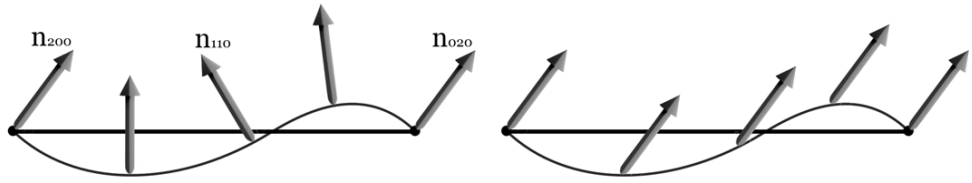
Rozkład punktów kontrolnych (w tym wypadku w postaci wektorów normalnych) na powierzchni trójkąta prezentuje się jak na rysunku 5.



Rys.5. Rozkład punktów kontrolnych wektorów normalnych.

Wykorzystanie funkcji wielomianowej drugiego stopnia do opisu wektorów normalnych, pomimo zastosowania kubicznej w przypadku geometrii, może budzić wątpliwości. Równania trzeciego stopnia mogą mieć trzy pierwiastki, drugiego tylko dwa. Można więc pomyśleć, że tracona jest część informacji o kształcie powierzchni. Otóż niekoniecznie musi tak być. Wszystko zależy od punktów kontrolnych wybranych dla wzoru 3.5. Rysunek 6 obrazuje hipotetyczną sytuację dla krawędzi, dla której wektory normalne na jej krańcach skierowane są w tym samym kierunku. Krzywa powstała w wyniku manipulacji punktami kontrolnymi geometrii ma kształt sinusoidalny dzięki zastosowaniu wielomianu trzeciego stopnia – posiada cztery punkty kontrolne (dwa wewnętrzne sterują kształtem). Prosta interpolacja wektorów normalnych spowodowałaby nakierowanie wszystkich pośrednich wektorów na ten sam kierunek – czego chcemy uniknąć. Funkcja wielomianowa drugiego stopnia z trzema punktami kontrolnymi w tym jednym pośrednim, mimo że jest o stopień niższa pozwala dobrze odwzorować normalną

do powierzchni w momencie umiejscowienia pośredniego punktu kontrolnego (wektora normalnego) w miejscu przegięcia krzywej i nakierowaniu go prostopadle do powierzchni w tym miejscu.



Rys.6. Porównanie funkcji stosowanej przez PN Triangles do interpolacji wektorów normalnych (lewy obraz) z interpolacją liniową (prawy obraz) po krzywej trzeciego stopnia.

Powyższa koncepcja pozwala na bardzo bliskie odwzorowanie normalnych do powierzchni. PN Triangles opiera się na niej, a same punkty kontrolne uzyskuje się jak poniżej:

$$
\begin{aligned}
n_{200} &= N_1 \quad n_{020} = N_2 \quad n_{002} = N_3 \\
r_{ij} &= 2\frac{(P_j - P_i)\cdot(N_i + N_j)}{(P_j - P_i)\cdot(P_j - P_i)} \\
h_{110} &= N_1 + N_2 - r_{12}(P_2 - P_1) \\
h_{011} &= N_2 + N_3 - r_{23}(P_3 - P_2) \\
h_{101} &= N_3 + N_1 - r_{31}(P_1 - P_3) \\
n_{110} &= h_{110} / \| h_{110} \| \\
n_{011} &= h_{011} / \| h_{011} \| \\
n_{101} &= h_{101} / \| h_{101} \|
\end{aligned}
\tag{3.6}
$$

gdzie: P – współrzędne wierzchołka,
N – wektor normalny wierzchołka.

Posiadając powyższe punkty kontrolne dla ewaluacji wektorów normalnych dla całej powierzchni, można dokonywać obliczeń oświetlenia z wygładzonymi przejściami między kolejnymi trójkątami. Owe gładkie przejścia zapewniają wektory normalne skierowane w tych samych kierunkach dla nakładających się wierzchołków z sąsiednich trójkątów.

3.1.3. Implementacja

Posiadając wiedzę o zasadzie działania techniki i dysponując odpowiednimi wzorami można przejść do implementacji. Istnieje kilka sposobów pozwalających uzyskać ten sam efekt. Poniżej zaprezentowany zostanie jeden, zapewniający optymalne rozłożenie obliczeń na wszystkie wywołania shadera kontroli teselacji. Na początku trzeba pamiętać o dostosowaniu aplikacji pod nowe API oraz ustawieniu liczby wierzchołków w płacie na trzy.

Listing 3.1. PN Triangles w shaderze kontroli teselacji (GLSL).

```
struct ControlPoints
{
	vec3 Pos[3];
	vec3 Nor[2];
};
layout(vertices = 3) out;
in vec3 inPos[];
in vec3 inNor[];
out ControlPoints CP[];
patch out vec3 center;
uniform float TessLevelInner;
uniform float TessLevelOuter;
#define ID gl_InvocationID

void main()
{
    int IDnext = (ID + 1) % 3;
    vec3 P1 = inPos[ID],      N1 = inNor[ID];
    vec3 P2 = inPos[IDnext],  N2 = inNor[IDnext];

    CP[ID].Pos[0] = P1;
    CP[ID].Pos[1] = (2 * P1 + P2 - dot(P2 - P1, N1) * N1)/3;
    CP[ID].Pos[2] = (2 * P2 + P1 - dot(P1 - P2, N2) * N2)/3;

    float r12 = 2 * dot(P2 - P1, N1 + N2) / dot(P2 - P1, P2 - P1);
    CP[ID].Nor[0] = N1;
    CP[ID].Nor[1] = N1 + N2 - r12 * (P2 - P1);

    gl_TessLevelOuter[ID] = TessLevelOuter;
    barrier();
    if (ID == 0)
    {
        vec3 E = (CP[0].Pos[1] + CP[0].Pos[2] + CP[1].Pos[1] +
                  CP[1].Pos[2] + CP[2].Pos[1] + CP[2].Pos[2]) / 6;
        vec3 V = (CP[0].Pos[0] + CP[1].Pos[0] + CP[2].Pos[0]) / 3;
        center = E + (E - V) / 2;
        gl_TessLevelInner[0] = TessLevelInner;
    }
}
```

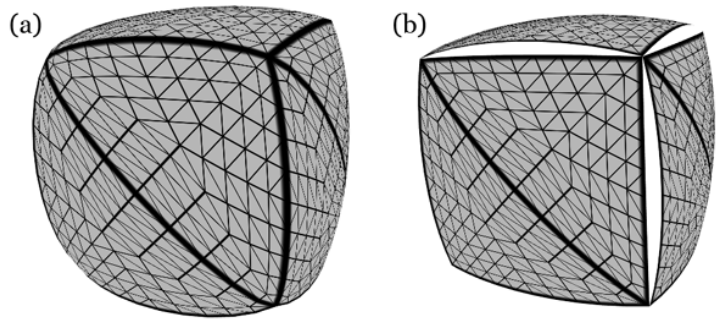
Listing 3.2. PN Triangles w shaderze ewaluacji teselacji (GLSL).

```
struct ControlPoints
{
	vec3 Pos[3];
	vec3 Nor[2];
};
layout(triangles, equal_spacing, ccw) in;
uniform mat4 MProjection;
uniform mat4 MModelview;
uniform mat3 MNormal;
```

```
in ControlPoints CP[];
patch in vec3 center;
out vec4 P;
out vec3 N;
out vec3 E;

void main()
{
    float u = gl_TessCoord.x, v = gl_TessCoord.y, w = gl_TessCoord.z;

    vec3 pos = CP[0].Pos[0]*w*w*w + CP[1].Pos[0]*u*u*u +
               CP[2].Pos[0]*v*v*v + CP[0].Pos[1]*w*w*u*3 +
               CP[0].Pos[2]*w*u*u*3 + CP[1].Pos[1]*u*u*v*3 +
               CP[1].Pos[2]*u*v*v*3 + CP[2].Pos[1]*v*v*w*3 +
               CP[2].Pos[2]*v*w*w*3 + center*u*v*w*6;

    vec3 nor = CP[0].Nor[0]*w*w + CP[1].Nor[0]*u*u + CP[2].Nor[0]*v*v +
               CP[0].Nor[1]*w*u + CP[1].Nor[1]*u*v + CP[2].Nor[1]*v*w;

    P = MProjection * MModelview * vec4(pos, 1);
    E = -(MModelview * vec4(pos,1)).xyz;
    N = MNormal * nor;
}
```

3.1.4. Podsumowanie metody

PN Triangles jest metodą, która bardzo dobrze wygładza siatkę modelu zależnie od nachylenia wektorów normalnych. Doskonale nadaje się do wygładzania m.in. sylwetki postaci. W przypadku bardziej kanciastych elementów, modele zdają się być jakby napchane (np. podeszwa buta będzie wypchnięta w dół). Wiąże się to z poważną wadą metody, która w przypadku różnych wektorów normalnych dla stykających się krawędzi powoduje ich rozszczepienie. Rysunek 7 prezentują taką sytuację.

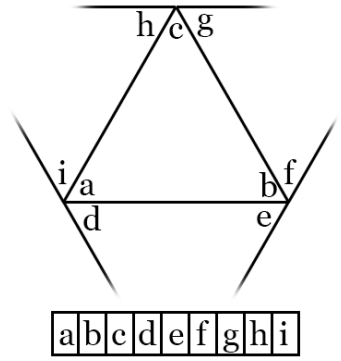


Rys.7. Sześcian z zastosowaną techniką PN Triangles. Wymagane są jednakowe normalne na krawędziach dwóch sąsiednich trójkątów; (a) – jednakowe normalne na wierzchołkach, (b) – różne normalne na wierzchołkach (bardziej prostopadłe do powierzchni).

3.2. PN-AEN Triangles

Technika PN-AEN Triangles [2] (właściwie ang. *Crack-Free Point-Normal Triangles using Adjacent Edge Normals*) jest – jak nazwa sugeruje – pewnym uzupełnieniem techniki PN Triangles. Powstała w odpowiedzi na poważną wadę poprzedniej metody, która w przypadku dwóch sąsiednich trójkątów złączonych krawędziami powoduje ich rozłączenie, gdy wektory normalne na nakładających się wierzchołkach są skierowane w różnych kierunkach.

Jako że technika jest rozwinięciem poprzedniej metody, koncepcja pozostaje ta sama. Nadal oblicza się te same punkty kontrolne, z tą różnicą, że już nie wyłącznie na podstawie informacji o wierzchołkach danego trójkąta, ale biorąc również pod uwagę wierzchołki wchodzące w skład trzech przylegających krawędzi sąsiednich trójkątów. Taka sytuacja wymaga modyfikacji w tablicy indeksów. Zamiast trzech wierzchołków na płat, konieczne jest przesłanie aż dziewięciu. Tablica indeksów rośnie zatem trzykrotnie. Sposób przechowywania informacji o kolejnych wierzchołkach przedstawia rysunek 8.



Rys.8. Kolejność przechowywanych indeksów wierzchołków w metodzie PN-AEN Triangles.

3.2.1. Zmiany w obliczeniach w stosunku do PN Triangles

Jak można wywnioskować ze wzoru 3.3, do obliczeń punktów kontrolnych znajdujących się na jednej krawędzi wystarczy informacja o dwóch wierzchołkach ją tworzących. Posiadając zatem informację o sąsiednich krawędziach można przewidzieć (policzyć), jak będą wyglądały ich punkty kontrolne. Na każdej z krawędzi znajdują się cztery punkty kontrolne, z których dwa znajdujące się na krańcach to wierzchołki trójkąta. Dwa wewnętrzne obliczane są wg wzoru 3.3. Obliczenia te dokonuje się dla właściwej krawędzi oraz krawędzi będącej w jej bezpośrednim sąsiedztwie (której informacja jest dostępna dzięki zmodyfikowanej tablicy indeksów). Otrzymane wyniki dla odpowiadających sobie punktów sumuje się i uśrednia jak pokazano we wzorze 3.7.

$$\begin{aligned} b_{210} &= (b^{w}{}_{210} + b^{z}{}_{210})/2 \qquad b_{120} = (b^{w}{}_{120} + b^{z}{}_{120})/2 \\ b_{021} &= (b^{w}{}_{021} + b^{z}{}_{021})/2 \qquad b_{012} = (b^{w}{}_{012} + b^{z}{}_{012})/2 \\ b_{102} &= (b^{w}{}_{102} + b^{z}{}_{102})/2 \qquad b_{201} = (b^{w}{}_{201} + b^{z}{}_{201})/2 \end{aligned} \tag{3.7}$$

gdzie: $b^{w}{}_{ijk}$ – wewnętrzne punkty kontrolne krawędzi (właściwego trójkąta),
$b^{z}{}_{ijk}$ – odpowiadające wewnętrzne punkty kontrolne sąsiedniej krawędzi.

Dzięki takiemu zabiegowi punkty kontrolne na krawędziach, dla przyległych sobie trójkątów, będą jednakowe bez względu na ułożenie wektorów normalnych. Należy przy tym pamiętać, by powyższych obliczeń nie stosować dla punktów kontrolnych wektorów normalnych, gdyż spowoduje to ich uśrednienie na krawędziach, czego wynikiem będzie niewłaściwe oświetlenie. Zależy nam na uniknięciu rozłączeń, jednak z zachowaniem właściwych normalnych do powierzchni. Zatem wzór 3.6 może pozostać bez zmian choć można się spodziewać drobnych przekłamań spowodowanych przesunięciem punktów kontrolnych siatki.

3.2.2. Implementacja

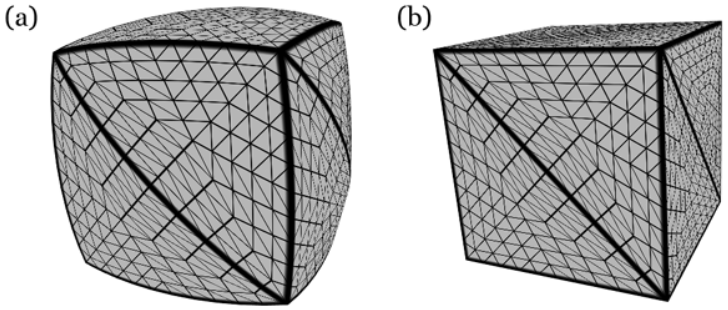
Ponieważ omawiana technika jest rozwinięciem koncepcji PN Triangles, dotychczasowe shadery nie wymagają większych zmian. Aktualizacji trzeba dokonać jedynie w głównej funkcji shadera kontroli teselacji, gdzie należy policzyć dodatkowe punkty kontrolne sąsiednich krawędzi. Ponadto, po zmodyfikowaniu tablicy indeksów, trzeba pamiętać o ustawieniu liczby wierzchołków w płacie na dziewięć.

Listing 3.3. PN-AEN Triangles w shaderze kontroli teselacji (GLSL).

```
//..
#define ID gl_InvocationID
void main()
{
    int IDnext    = (ID + 1) % 3;   // następny wierzchołek
    int IDadd     = (ID * 2) + 3;   // sąsiad wierzchołka ID
    int IDaddNext = IDadd + 1;      // sąsiad wierzchołka IDnext

    vec3 P1i = inPos[ID],        N1i = inNor[ID];
    vec3 P1o = inPos[IDadd],     N1o = inNor[IDadd];
    vec3 P2i = inPos[IDnext],    N2i = inNor[IDnext];
    vec3 P2o = inPos[IDaddNext], N2o = inNor[IDaddNext];

    vec3 posIns, posOut;

    CP[ID].Pos[0] = P1i;
    posIns = (2 * P1i + P2i - dot(P2i - P1i, N1i) * N1i) / 3;
    posOut = (2 * P1o + P2o - dot(P2o - P1o, N1o) * N1o) / 3;
    CP[ID].Pos[1] = (posIns + posOut) / 2;

    posIns = (2 * P2i + P1i - dot(P1i - P2i, N2i) * N2i) / 3;
    posOut = (2 * P2o + P1o - dot(P1o - P2o, N2o) * N2o) / 3;
    CP[ID].Pos[2] = (posIns + posOut) / 2;
    // centrum i punkty kontrolne wektorów normalnych oblicza się tak jak
    // w PN Triangles
}
```

3.2.3. Podsumowanie metody

PN-AEN Triangles jest skutecznym rozszerzeniem metody PN Triangles. Nie tylko zapobiega powstawaniu rozłączeń na krawędziach, ale też zachowuje ostre krawędzie. Wygładza zatem model tam gdzie jest to rzeczywiście konieczne. Minusem metody jest konieczność przechowywania informacji o sąsiednich krawędziach w tablicy indeksów. Rysunek 9 uwidacznia zmiany w stosunku do źródłowej techniki.

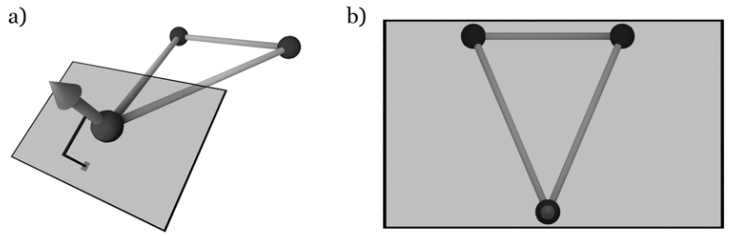


Rys.9. (a) – odpowiada rysunkowi 7b przy zastosowaniu techniki PN-AEN Triangles, (b) – wektory normalne skierowane są prostopadle do powierzchni.

3.3. Teselacja Phonga

Cieniowanie Phonga [4] jest powszechnie znaną techniką polegającą na interpolacji wektorów normalnych na powierzchni wielokąta w celu obliczeń oświetlenia z dokładnością do pojedynczych pikseli. Teselacja Phonga [5] jest niejako przeniesieniem owej koncepcji do zastosowań wygładzania steselowanej siatki wielokątów. Podobnie jak w przypadku PN Triangles nie wymaga żadnej dodatkowej informacji poza pozycją oraz wektorami normalnymi wierzchołków trójkąta. Jest ponadto mniej złożona obliczeniowo, co niestety implikuje mniejszą dokładność.

Metoda ta jest stosunkowo łatwa w implementacji, z racji niewielkiej ilości obliczeń. Mimo to zasada działania nie jest tak prosta jak w cieniowaniu Phonga. W ogólności można ją wyrazić jednym zdaniem: Jest to barycentryczna interpolacja między trzema trójkątami ortogonalnie zrzutowanymi na płaszczyzny styczne określone przez wektory normalne wierzchołków. Łatwiej zrozumieć powyższą definicję analizując ją od końca. Rysunek 10a przedstawia trójkąt, gdzie przy jednym wierzchołku narysowano wektor normalny. Płaszczyzną styczną będzie płaszczyzna, której powierzchnia jest prostopadła do tego wektora i która zaczepiona jest u jego podstawy. Wykonanie ortogonalnej projekcji trójkąta polega na jego prostopadłym rzucie na płaszczyznę styczną (rys. 10b). Posiadając trzy w ten sposób zrzutowane trójkąty (po jednym na wierzchołek), na każdym z nich wyznacza się punkt określony przez współrzędne barycentryczne. Trzy powstałe w ten sposób punkty tworzą trójkąt, którego te same współrzędne barycentryczne definiują punkt leżący na szukanej powierzchni. Interpolacja barycentryczna współrzędnych spowoduje uformowanie kompletnego kształtu powierzchni.



Rys.10. (a) – płaszczyzna styczna do wektora normalnego, (b) – ortogonalny rzut trójkąta na płaszczyznę styczną.

3.3.1. Obliczenia

Wymagane obliczenia, jak już było to wspomniane, są stosunkowo proste. Całość opiera się na dwóch wzorach, z których pierwszy definiuje ortogonalną projekcję trójkątów na płaszczyznę styczną.

$$\pi_i(q) = q - ((q - P_i) \cdot N_i) N_i \tag{3.8}$$

gdzie: $\pi_i(q)$ – ortogonalna projekcja punktu q na i-tą płaszczyznę styczną,
P_i – współrzędne i-tego wierzchołka trójkąta,
N_i – wektor normalny i-tego wierzchołka trójkąta.

Drugi natomiast decyduje o położeniu punktu na powierzchni przy zadanych współrzędnych barycentrycznych.

$$p(u,v,w) = \begin{bmatrix} u & v & w \end{bmatrix} \begin{bmatrix} \pi_i(P(u,v,w)) \\ \pi_j(P(u,v,w)) \\ \pi_k(P(u,v,w)) \end{bmatrix} \tag{3.9}$$

gdzie: $P(u,v,w)$ – zinterpolowany punkt wewnątrz trójkąta – $(u,v,w)(P_i,P_j,P_k)^T$,

Teselacja Phonga w naturalnej formie powoduje znacznie mocniejsze odkształcenia siatki w porównaniu do metody PN Triangles. W celu ich ograniczenia można zastosować współczynnik określający w jakim stopniu metoda ma być aktywna. Dobre rezultaty otrzymuje się przy wartości 0.75. Wzór 3.9 uwzględniający owy współczynnik prezentuje się następująco:

$$p(u,v,w) = (1-\alpha)P(u,v,w) + \alpha \begin{bmatrix} u & v & w \end{bmatrix} \begin{bmatrix} \pi_i(P(u,v,w)) \\ \pi_j(P(u,v,w)) \\ \pi_k(P(u,v,w)) \end{bmatrix} \tag{3.10}$$

gdzie: α – współczynnik intensywności (0-brak wygładzania, 1-maksymalne wygładzenie).

Po rozwinięciu wzoru 3.9 otrzymujemy funkcję wielomianową drugiego stopnia (płat drugiego stopnia), której postać pozwala na łatwą implementację w programie.

$$\begin{aligned} p(u,v,w) = {} & u^2 P_i + v^2 P_j + w^2 P_k + \\ & uv(\pi_i(P_j) + \pi_j(P_i)) + \\ & vw(\pi_j(P_k) + \pi_k(P_j)) + \\ & wu(\pi_k(P_i) + \pi_i(P_k)) \end{aligned} \tag{3.11}$$

gdzie: P_i – współrzędne i-tego wierzchołka trójkąta.

Warto zauważyć, że Teselacja Phonga z powodu zastosowania wielomianu drugiego stopnia nie jest w stanie odwzorować poprawnie powierzchni, gdy wektory normalne nachylone są w tym samym kierunku. Oznacza to że nie jest w stanie symulować zmian wypukłości w przeciwieństwie do techniki PN Triangles.

3.3.2. Wektory normalne w teselacji Phonga

Teselacja Phonga nie wprowadza nic nowego w kwestii ewaluacji wektorów normalnych na steselowanej powierzchni. W tym przypadku wykorzystany jest schemat z cieniowania Phonga czyli prosta interpolacja liniowa. Jako że do opisu geometrii wykorzystana jest funkcja wielomianowa drugiego stopnia, nie ma potrzeby stosować podobnej również do obliczeń wektorów normalnych, gdyż w geometrii i tak nie wystąpią zmiany wypukłości.

3.3.3. Implementacja

Fakt, iż Teselacja Phonga również korzysta wyłącznie z informacji o pozycji wierzchołków oraz ich wektorach normalnych powoduje, że nie potrzeba wykonywać żadnych zmian w silniku graficznym względem techniki PN Triangles. Aktualizacji wymagają jedynie shadery kontroli i ewaluacji teselacji.

Listing 3.4. Teselacja Phonga w shaderze kontroli teselacji (GLSL).

```
struct TangentTriangle
{
    vec3 PI[3];
    vec3 Nor;
};
layout(vertices = 3) out;
in vec3 inPos[];
in vec3 inNor[];
out TangentTriangle T[];
uniform float TessLevelInner;
uniform float TessLevelOuter;

#define ID gl_InvocationID

void main()
{
    int IDnext1 = (ID + 1) % 3;
    int IDnext2 = (ID + 2) % 3;
    vec3 P0 = inPos[ID];
    vec3 P1 = inPos[IDnext1];
    vec3 P2 = inPos[IDnext2];
    vec3 N0 = inNor[ID];

    T[ID].PI[0] = P0;
    T[ID].PI[1] = P1 - dot(P1 - P0, N0) * N0;
    T[ID].PI[2] = P2 - dot(P2 - P0, N0) * N0;
    T[ID].Nor   = N0;

    gl_TessLevelOuter[ID] = TessLevelOuter;
    if (ID == 0)
        gl_TessLevelInner[0] = TessLevelInner;
}
```

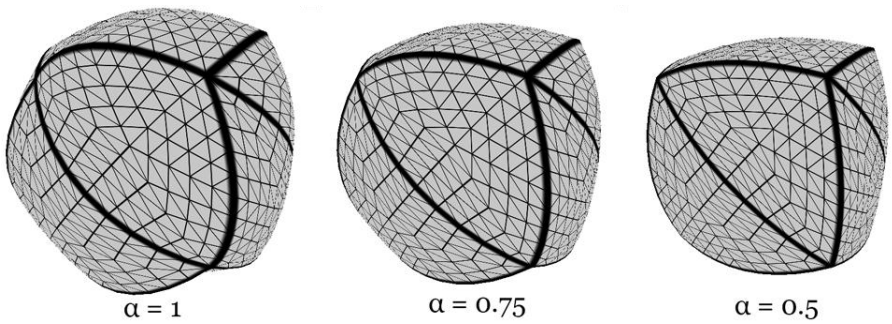
Listing 3.5. Teselacja Phonga w shaderze ewaluacji teselacji (GLSL).

```
struct TangentTriangle
{
    vec3 PI[3];
    vec3 Nor;
};
layout(triangles, equal_spacing, ccw) in;
uniform float alpha;
in TangentTriangle T[];
//..
void main()
{
    float u = gl_TessCoord.x, v = gl_TessCoord.y, w = gl_TessCoord.z;

    vec3 pos1 = T[0].PI[0]*u + T[1].PI[0]*v + T[2].PI[0]*w;
    vec3 pos2 = T[0].PI[0]*u*u + T[1].PI[0]*v*v + T[2].PI[0]*w*w +
                (T[0].PI[1] + T[1].PI[2])*u*v +
                (T[1].PI[1] + T[2].PI[2])*v*w +
                (T[2].PI[1] + T[0].PI[2])*w*u;
    vec3 pos = (1 - alpha)*pos1 + alpha*pos2;
    vec3 nor = T[0].Nor*u + T[1].Nor*v + T[2].Nor*w;
    //..
}
```

3.3.5. Podsumowanie metody

Powyższa metoda jest ciekawą alternatywą dla PN Triangles, głównie z racji niewielkiego kosztu obliczeniowego potrzebnego do uzyskania zbliżonego efektu. Jej głównym minusem, podobnie jak PN Triangles, jest występowanie rozłączeń w przypadku niejednorodnych wektorów normalnych na wierzchołkach. Jednym z rozwiązań jest utworzenie specjalnego kanału do przechowywania uśrednionych wektorów normalnych na potrzeby teselacji. Niestety rodzi to kolejny problem, gdyż w niektórych przypadkach obiekty mogą wyglądać jakby wypchane. Choć efekt ten można wyeliminować, poprzez dołożenie dodatkowych niewielkich wielokątów w pobliżu ostrych krawędzi, to zabieg ten wymaga już interwencji grafika.

W sytuacji, gdy jakość nie jest przeważającym czynnikiem, metoda ta jest dobrym zamiennikiem dla PN Triangles. Rysunek 11 prezentuje efekt zastosowania Teselacji Phonga dla różnych współczynników intensywności wygładzania.



Rys.11. Teselacja Phonga dla różnych współczynników intensywności wygładzania.

4. ODWZOROWANIE PRZEMIESZCZEŃ

Techniki wygładzania siatki w połączeniu ze sprzętową teselacją pozwalają w efektywny sposób pozbyć się kanciastości na modelach trójwymiarowych. Wygładzanie to jednak nie jedyne zastosowanie dla teselacji. Jednym z coraz popularniejszych efektów stosowanych w grach jest odwzorowanie przemieszczeń (ang. *Displacement Mapping*). Technika ta polega, jak sama nazwa sugeruje, na przemieszczaniu wierzchołków modelu. Informacja o kierunku i stopniu przemieszczenia przechowywana jest w teksturze o odcieniach szarości zwanej mapą przemieszczeń (ang. *displacement map*). Efekt jest tym lepszy im więcej wierzchołków opisuje dany model. Teselacja jest więc idealnym partnerem omawianej techniki.

4.1. Implementacja w nowym potoku renderującym

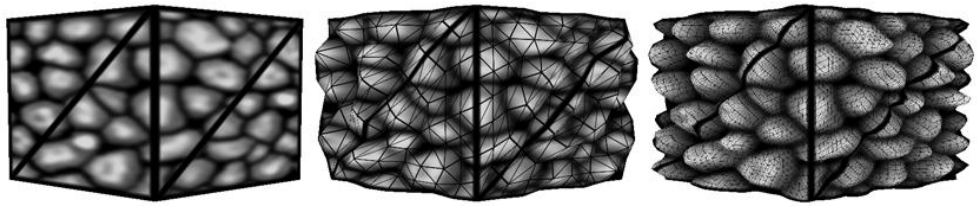
Odwzorowanie przemieszczeń uzyskuje się w shaderze ewaluacji teselacji próbkując teksturę w miejscu powstawania nowych wierzchołków. Uzyskaną wartość wykorzystuje się w obliczeniach przemieszczenia, którego kierunek wyznacza wektor normalny. Należy pamiętać, by podczas tworzenia mapy przemieszczeń określić umowny poziom zerowy, niepowodujący przekształceń. Najbardziej optymalnym rozwiązaniem jest ustawienie tej wartości na 0.5 w skali [0-1] (127 w skali [0-255]). Pozwala to na przemieszczanie wierzchołków zarówno na zewnątrz jak i w głąb modelu w tym samym stopniu. Ewentualne zmiany tego poziomu można w niektórych przypadkach regulować bezpośrednio w shaderze.

Odwzorowanie przemieszczeń często wykonuje się na „płaskiej" teselacji, bez udziału technik wygładzania siatki (nic nie stoi jednak na przeszkodzie by je zastosować). Przykład implementacji omawianej techniki w shaderze ewaluacji teselacji przedstawia Listing 4.1.

Listing 4.1. Odwzorowanie przemieszczeń w shaderze ewaluacji teselacji (GLSL).

```
layout(triangles, equal_spacing, ccw) in;
uniform sampler2D dispMap;
in vec3 Pos[];
in vec3 Nor[];
in vec2 TexCoord[];
//...
void main()
{
    float u = gl_TessCoord.x, v = gl_TessCoord.y, w = gl_TessCoord.z;

    vec3 outPos = Pos[0]*u + Pos[1]*v + Pos[2]*w;
    vec3 outNor = Nor[0]*u + Nor[1]*v + Nor[2]*w;
    vec2 outTC  = TexCoord[0]*u + TexCoord[1]*v + TexCoord[2]*w;

    vec4 h = texture(dispMap, outTC); // wartość przemieszczenia
    outPos += (h.x - 0.5) * outNor;   // przemieszczenie wzdłuż wekt. nor.
    //...
}
```

Powyższy fragment kodu spowoduje przemieszczenie wierzchołków z jasnymi fragmentami tekstury na zewnątrz modelu, a z ciemnymi do wewnątrz jak na rysunku 12.



Rys.12. Odwzorowanie przemieszczeń dla różnych stopni teselacji. Zaczynając od lewej strony – brak teselacji, teselacja o współczynniku 8, teselacja o współczynniku 32.

4.2. Uzyskiwanie mapy przemieszczeń

Fizyczne przemieszczenie wierzchołków, będące efektem odwzorowania przemieszczeń, wywołuje problemy w symulacjach oświetlenia. Zmieniony kształt powierzchni wymaga zaktualizowania wektorów normalnych. Istnieją metody pozwalające określić wektor normalny na podstawie map wysokości bezpośrednio w programach cieniujących [6], jednak jest to dodatkowy nakład obliczeń. Problem można rozwiązać na etapie modelowania geometrii, wykorzystując model wysokiej rozdzielczości (ang. *high poly*) do utworzenia zarówno mapy normalnych (ang. *normal map*), przechowującej informacje o wektorach normalnych, jak i mapy przemieszczeń. Wiele programów do modelowania dysponuje odpowiednimi narzędziami pozwalającymi „wypalać" szczegóły geometrii na teksturze. Zaleca się stosowanie tego samego narzędzia do uzyskania zarówno mapy normalnych jak i mapy przemieszczeń, by zachować jak najdokładniejsze odwzorowanie.

Istnieją również programy, które potrafią generować mapy normalnych oraz mapy przemieszczeń na podstawie tekstur. Można do nich zaliczyć m.in. CrazyBump czy PixPlant. Dostępna jest również wtyczka autorstwa Nvidii do programu Photoshop o nazwie Normal Map Filter, która umożliwia utworzenie mapy wysokości na podstawie mapy normalnych, jednak nie zawsze pozwala uzyskać efekt zgodny z oczekiwaniami.

4.3. Odwzorowanie przemieszczeń a techniki symulowania wypukłości na płaskich powierzchniach

Omawiana technika jest sposobem na symulowanie wypukłości za pomocą fizycznej manipulacji geometrią. Przed wprowadzeniem teselacji powstało jednak wiele technik odwzorowujących wypukłości na płaskich wielokątach, do których zaliczyć można (nazwy angielskie): *bump mapping*, *normal mapping*, *parallax mapping*, *parallax occlusion mapping*, *relief mapping*, *cone step mapping* czy *relaxed cone stepping*. Powstaje zatem pytanie czy owe techniki przestają mieć praktyczne zastosowanie po wprowadzeniu teselacji. Odpowiedź nie jest oczywista i zależy od wielu czynników. Każda z wyżej wymienionych metod ma swoje wady jak i zalety, jednak najważniejszą różnicą w stosunku do odwzorowania przemieszczeń jest fakt, że paradoksalnie nie potrafią one symulować wypukłości. Techniki te wykorzystują różne mechanizmy oparte na manipulacji wektorami normalnymi oraz współrzędnymi tekstury, jednak nie są w stanie zmienić koloru fragmentu, który znajduje się poza opisywanym wielokątem. Wszystkie widoczne wypukłości są efektem niejako wklęśnięć odpowiednich fragmentów.

Zaletą odwzorowania przemieszczeń jest zatem możliwość rzeczywistego, fizycznego symulowania wypukłości. Trzeba jednak zauważyć, że aby jakość odkształceń była zadowalająca i szczegółowa trzeba dostatecznie mocno zagęścić siatkę powierzchni. Koszt związany z tak gęstą teselacją może przewyższać akceptowalne poziomy spadku

wydajności. W takim wypadku dobrze jest mieć alternatywę w postaci technik symulowania wypukłości na płaskich powierzchniach. Poza tym w technikach tych, jakość uzyskiwanych wypukłości jest niezależna od gęstości geometrii (ale zależna od rozdzielczości tekstur przez nie stosowanych), więc kanciastości w nich zwykle nie występują.

Technika odwzorowania normalnych jest przydatnym, a wręcz niezbędnym narzędziem podczas korzystania z odwzorowania przemieszczeń, gdyż wykorzystuje wspomnianą wcześniej mapę normalnych do poprawnych obliczeń oświetlenia. Dzięki jej zastosowaniu model, nawet w przypadku przejścia w stan nieaktywnej teselacji, nadal dysponuje detalami widocznymi przy odpowiednim oświetleniu.

W przypadku ciągłych płaskich powierzchni (bez ostrych krawędzi) warto zastosować popularny *parallax occlusion mapping*, który mimo że jest kosztowny, to uzyskany efekt niczym nie ustępuje gęstemu odwzorowaniu przemieszczeń.

5. PODSUMOWANIE

Wszystkie opisane w artykule techniki są współcześnie stosowane w realistycznych grach komputerowych oraz silnikach gier (m.in. UDK, CryEngine). Teselacja jest narzędziem, które pozwala zapomnieć o ograniczeniach gęstości podstawowej siatki modeli. Oczywiście korzystanie z niej nie jest darmowe, gdyż wiąże się z koniecznością wykonania trzech dodatkowych kroków w potoku renderującym. Te jednak potrafią przenieść wiele gier na wyższy poziom realizmu.

BIBLIOGRAFIA

[1] C. Loop, S. Schaefer: *Approximating Catmull-Clark subdivision surfaces with bicubic patches*, ACM Transactions on Graphics (TOG), 2008.
[2] J. McDonald, M. Kilgard: *Crack-Free Point-Normal Triangles using Adjacent Edge Normals*, Nvidia, 2010.
[3] A. Vlachos, J. Peters, C. Boyd, J. L. Mitchell: *Curved PN Triangles* w: Proceedings of the 2001 Symposium on Interactive 3D Graphics, s. 159-166, ACM Press, 2001.
[4] B. T. Phong, *Illumination for Computer Generated Pictures*, Communications of the ACM, 18(6), 1975.
[5] T. Boubekeur, M. Alexa: *Phong Tessellation* w: SIGGRAPH Asia, vol. 27, ACM Trans. Graphics, 2008.
[6] N. Tatarchuk: *Dynamic Terrain Rendering on GPUs Using Real-Time Tessellation* w: ShaderX7, s. 73-105, Charles River Media, 2009.

HARDWARE TESSELLATION TECHNIQUES FOR SMOOTHING AND DETAILING 3D MODELS

Summary

DirectX 11 and OpenGL 4 have introduced new rendering pipeline with highly anticipated tessellation. As time passes, more and more graphics engines are adapt to support this technology, so one cannot remain indifferent to it. In the following article overall concept behind tessellation will be presented, both from theoretical and practical sides. It will also include description of new rendering pipeline, along with dedicated tessellation shaders. The main part of the article will contain a description of the various techniques used to achieve effects such as smooth surfaces or displacement mapping, which allow for effective detailing three-dimensional models.