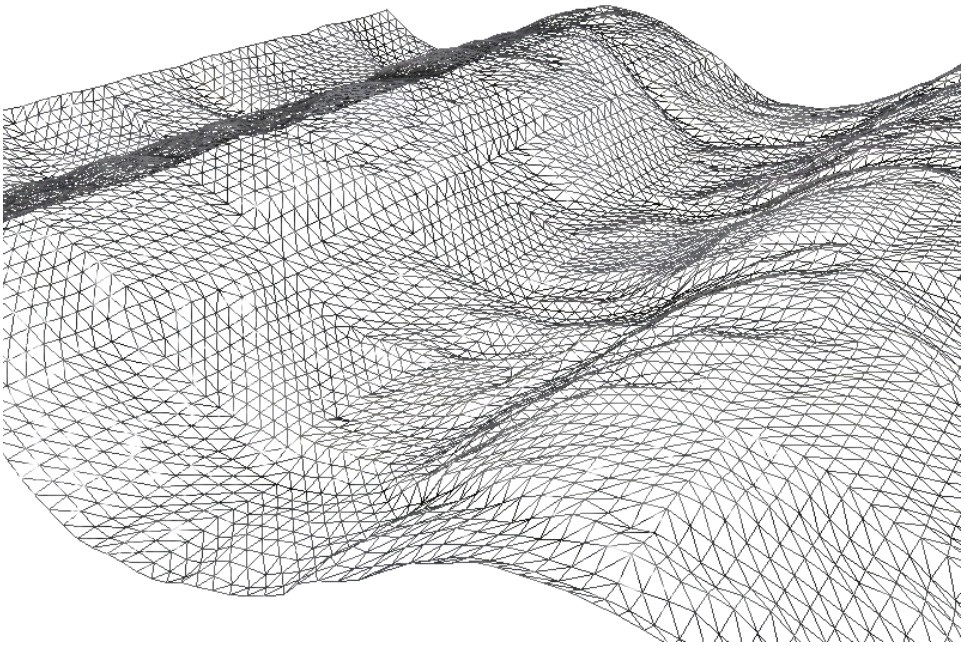
Projekt 4. Teselacja

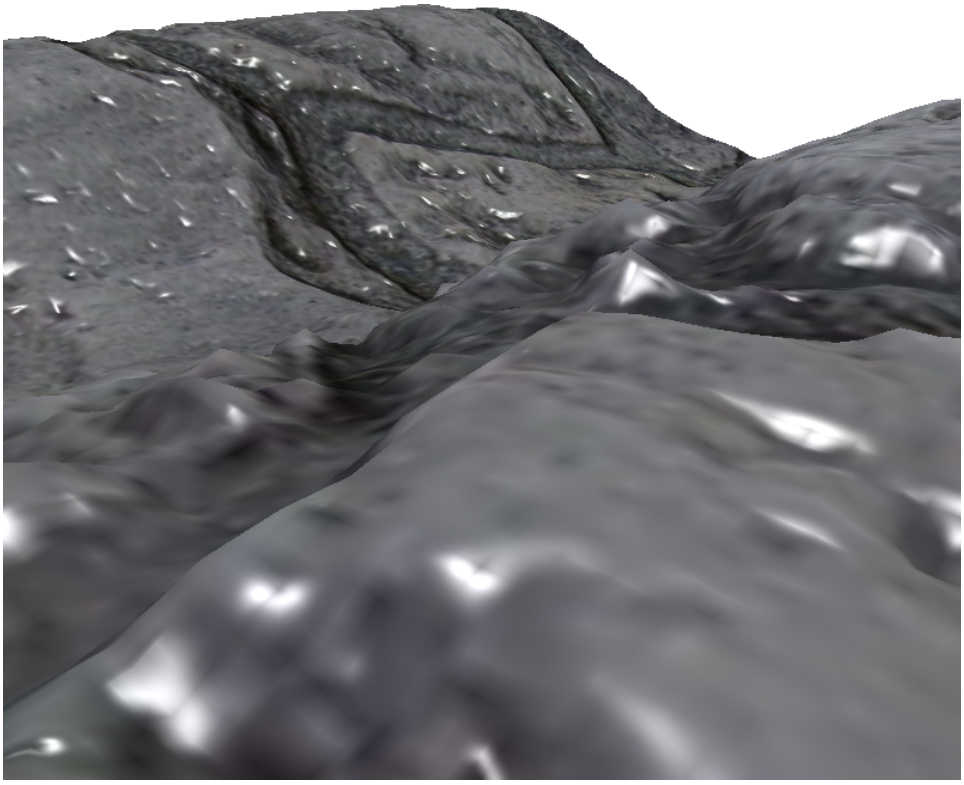
Paweł Aszklar
P.Aszklar@mini.pw.edu.pl

Warszawa, 21 maja 2021

1 Wstęp

1. Projekt jest indywidualny
2. Termin oddania projektu to 11. czerwca
3. Rozwiązanie należy zaprezentować na laboratoriach bądź przesłać spakowany kod źródłowy razem pozostałymi plikami, niezbędnymi do uruchomienia programu, na adres P.Aszklar@mini.pw.edu.pl.
4. Projekt można pisać w C# lub C++ korzystając z biblioteki DirectX lub OpenGL i języka shaderów HLSL lub GLSL.
5. Zadanie składa się z kilku części. Nie trzeba realizować wszystkich.





2 Opis zadania

2.1 Część I [1 pkt]

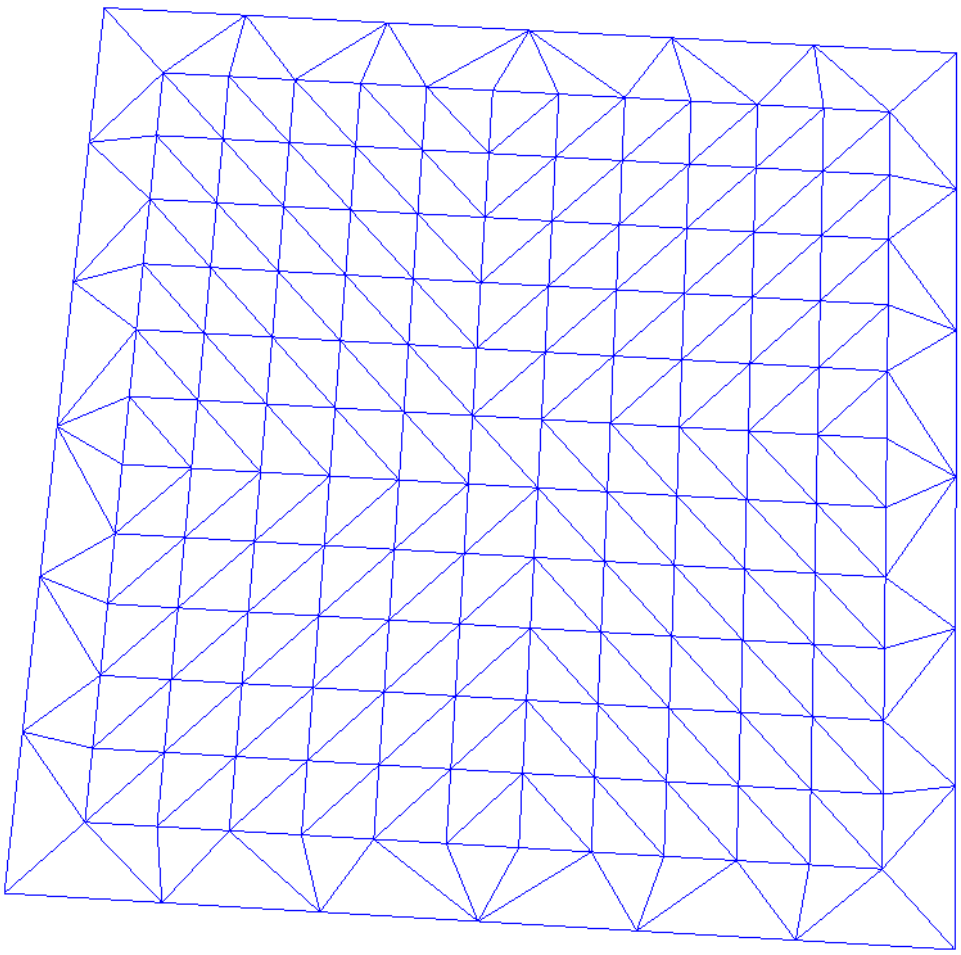
a) Teselacja powierzchni czworoboku. Program powinien wyświetlać czworobok opisany za pomocą 4 wierzchołków (stanowiących jego *punkty kontrolne*).

b) Należy wykorzystać dwa elementy programowalnego potoku renderowania do podziału powierzchni czworoboku na siatkę trójkątów. Te dwa nowe shadery to *shader powłoko* (*Hull Shader*) oraz *shader dziedziny* (*Domain Shader*). — W opisie zadania używana jest nomenklatura z biblioteki DirectX. W bibliotece OpenGL użycie teselacji przebiega w bardzo podobny sposób, jednak poszczególne etapy potoku renderowania znane są po innymi nazwami. *Hull Shader* z DirectX nazwany został *Tessellation Control Shader* natomiast *Domain Shader* jako *Tessellation Evaluation Shader*.

c) Podział powinien być sterowany za pomocą dwóch współczynników przekazywanych do shaderów przez program: współczynnika teselacji krawędzi oraz współczynnika teselacji wnętrza czworoboku. Program

powinien umożliwiać zmianę wartości tych współczynników w trakcie działania, za pomocą klawiatury.

d) Siatka powinna być wyświetlana ze stałym kolorem w trybie krawędziowym (rysowane powinny być tylko krawędzie trójkątów, bez ich wypełniania). Wyłączyć należy też obcinanie ścian tylnych.



2.2 Część II [+2 pkt]

a) Teselacja płatka Béziera. Tym razem generowana siatka opisana powinna być przez 16 punktów, będących punktami kontrolnymi prostokątnego płatka Béziera stopnia (3, 3)

b) Punkty wygenerowanej siatki leżeć powinny na powierzchni płatka. Do wyznaczenia ich współrzędnych można skorzystać bezpośrednio ze

wzoru:

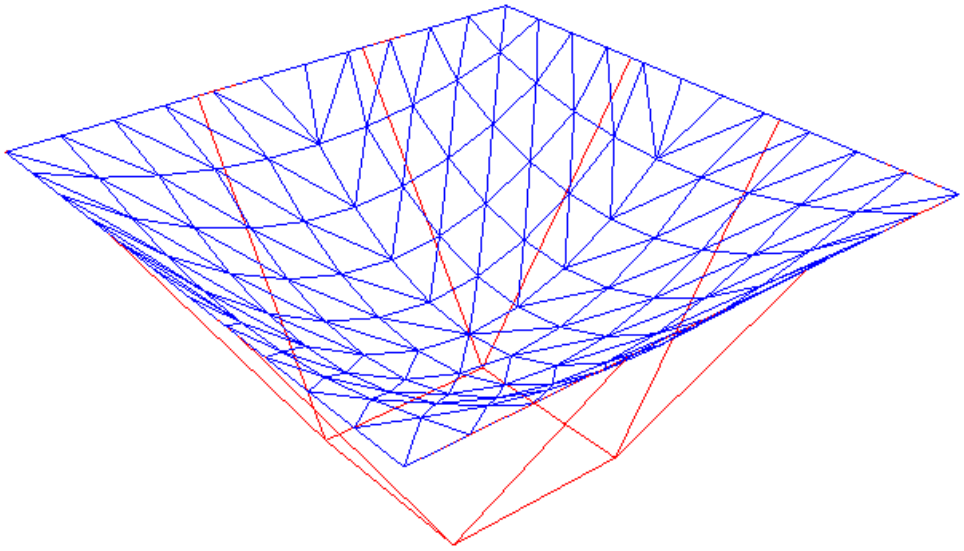
$$p(u,\, v) = \sum_{i=0}^{3} \left(\sum_{j=0}^{3} p_{ij} B_j^3(v) \right) B_i^3(u)$$

, gdzie

$$B_i^3(t) = \binom{3}{i} t^i (1-t)^{3-i}$$

c) Zdefiniuj przynajmniej dwa różne zestawy punktów kontrolnych, pomiędzy którymi można przełączać w trakcie działania programu.

d) Poza wygenerowaną siatką, program powinien mieć też możliwość wyświetlania siatki wielościanu kontrolnego danego płatka

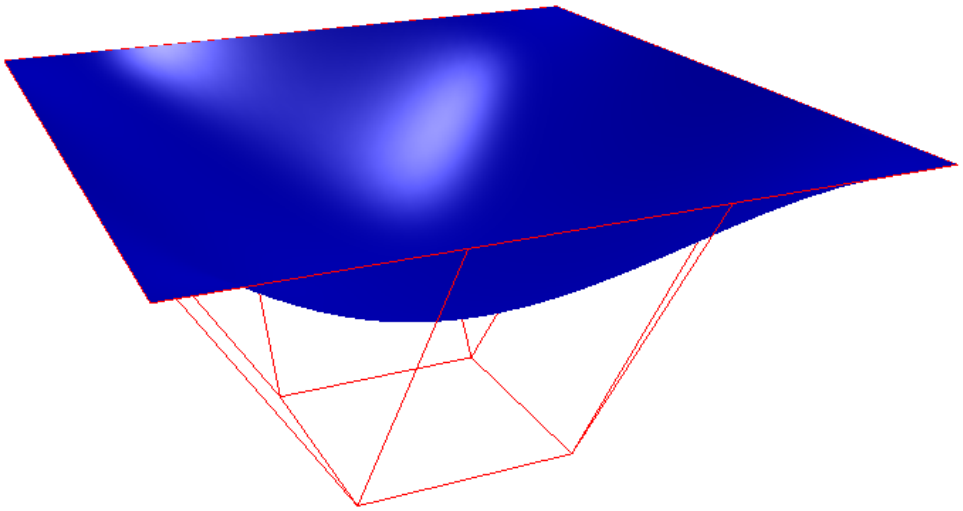


2.3 Część III [+2pkt]

a) Płatek powinien być wyświetlany tak jak w poprzedniej części, ale dodatkowo w każdym punkcie wygenerowanej siatki należy wyznaczyć wektor normalny do powierzchni danego płatka.

b) Trójkąty siatki powinny być rysowane wypełnione i pocieniowane zgodnie z modelem Phonga. Do sceny należy dodać pojedyncze punktowe źródło światła koloru białego.

c) Program powinien pozwalać na przełączanie pomiędzy rysowaniem pocieniowanej powierzchni oraz widokiem krawędziowym.



2.4 Część IV [+2.5 pkt]

a) Tym razem program powinien wyświetlać powierzchnię złożoną z siatki 4×4 płatków Béziera połączonych z ciągłością co najmniej C^0.

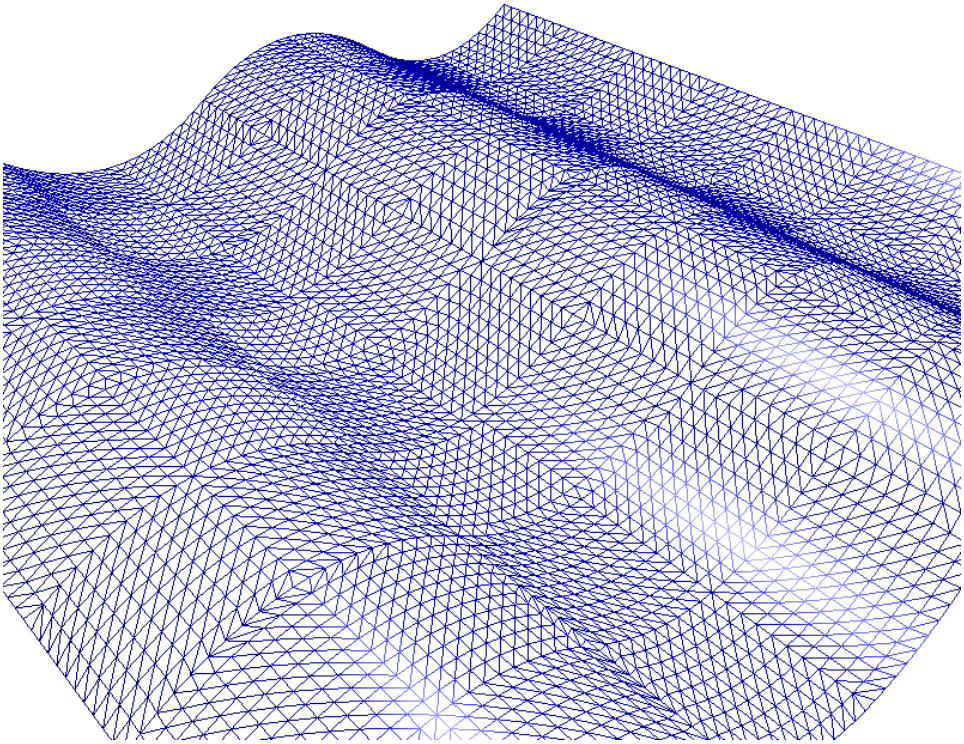
b) Zaimplementuj dynamicznie wyznaczany poziom szczegółowości (Level Of Detail) generowanej siatki w zależności od odległości płatka od kamery. Wartość współczynnika teselacji można wyznaczyć na podstawie wartości współrzędnej głębokości w układzie kamery (odległość od kamery) ze wzoru:

$$\text{factor}(z) = -16 * \log_{10}(z * 0.01)$$

Wzór ten zakłada, ze wartości współrzędnej głębokości należą do przedziału [0.01, 100].

c) Wygenerowana siatka nie powinna zawierać dziur. W tym celu należy zapewnić, że wspólne krawędzie sąsiadujących płatków ulegają podziałowi z tą samą wartością współczynnika teselacji.

d) Wartości współczynników teselacji, które do tej pory przekazywane były do shaderów powinny zostać użyte do modyfikacji wartości zwracanych przez funkcję factor, w efekcie globalnie zwiększając lub zmniejszając poziom szczegółowości sceny.



2.5 Część V [+2.5 pkt]

a) Dodaj do programu trzy tekstury: koloru, normalnych oraz przemieszczeń (wysokości), reprezentujących tą samą powierzchnię. Przykładowe tekstury pobrać można stąd.

b) Dla każdego wierzchołka wygenerowanej siatki należy pobrać wartość wysokości z tekstury przemieszczeń (*diplacement mapping*). Wartość ta będzie z przedziału [0, 1] i może wymagać przeskalowania.

c) Położenie wierzchołka należy przesunąć wzdłuż wektora normalnego w tym punkcie o wczytaną odległość.

d) Do pobierania wartości z tekstury w shaderze dziedziny (*Domain Shader*) należy użyć funkcji `SampleLevel`:

```
float h = map.SampleLevel(sampler, coords, mipLvl).x;
```

Jej trzecie parametr pozwala określić poziom mipmap, która zostanie użyta przy wczytywaniu z tekstury. Wartość tą musimy podać ręcznie, gdyż tylko w shaderze pikseli może być ona wyznaczona automatycznie.

e) Poziom mipmap można wyznaczyć używając z następującego wzoru (korzystającego ze zdefiniowanej wcześniej funkcji factor):

$$\text{mipLevel}(z) = 6 - \log_2(\text{factor}(z))$$

Poziom ten musi zależeć wyłącznie od współrzędnej z danego punktu, aby wierzchołki w różnych płatkach o tych samych współrzędnych otrzymały takie samo przesunięcie (inaczej siatka będzie niespójna).

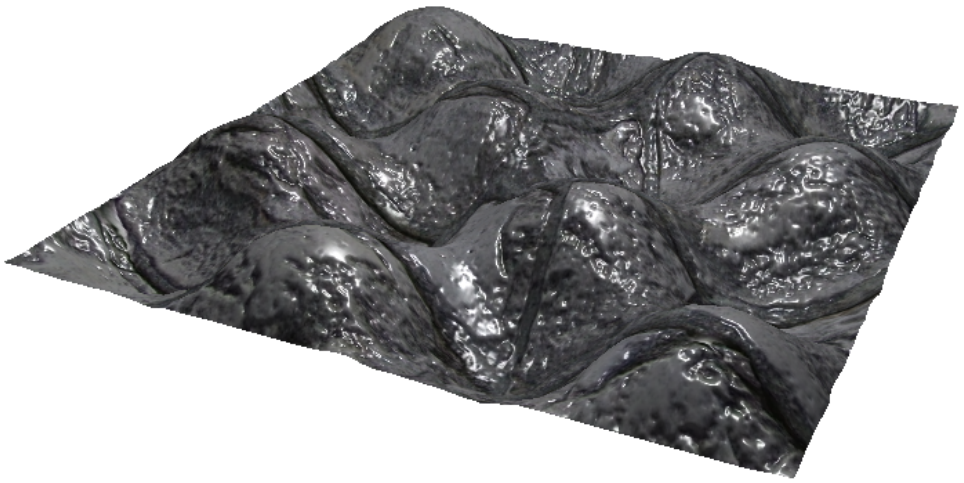
f) Współrzędne tekstury powinny być dobrane tak, aby wybrane tekstury rozciągnięte zostały bez zawijania na wszystkie 16 płatków.

g) Do shadera pikseli przekazać należy zarówno położenie, wektor normalny a także współrzędne tekstury, wektory styczny oraz binormalny.

h) Kolor powierzchni dla rysowanego piksela w shaderze pikseli pobrany powinien być z tekstury koloru, natomiast wektor normalny, z tekstury normalnych. W przykładowej teksturze wektory normalne zakodowane są w ten sam sposób, co w poprzednim projekcie. Zwrócić uwagę należy na fakt, że wektor normalny z tekstury określony jest w stycznym do powierzchni układzie lokalnym punktu (bazą tego układu są wektory styczny, binormalny i normalny otrzymane z wcześniejszych etapów potoku renderowania). Należy go przekształcić do układu globalnego (lub widoku) korzystając z poniższego wzoru:

$$texNorm.x * tangent + texNorm.y * binormal + texNorm.z * normal$$

Wektora otrzymanego w wyniku tej operacji należy użyć przy cieniowaniu powierzchni.



3 Program przykładowy

Przykładowy program pobrać można tutaj. Pozwala on zapoznać się z podstawami stosowania shaderów powłoki i dziedziny. Demonstruje on działanie potoku renderowania rozszerzonego o dwa dodatkowe shadery i stanowi dobry punkt startowy przy implementacji tego projektu. Poniżej znaleźć można opis co ciekawszych fragmentów programu. Więcej informacji teselacji w DirectX znaleźć można tutaj.

3.1 Wstęp

W programach korzystających z teselacji nie definiujemy rysowanych obiektów za pomocą siatek prostych kształtów geometrycznych takich jak punkty, odcinki, trójkąty, opisanych odpowiednio 1, 2 lub 3 wierzchołkami. Zamiast tego obiekty stworzone są z płatków opisanych za pomocą punktów kontrolnych, których znaczenie zależy zupełnie od nas. Płatki tworzyć może od 1 do 32 punktów kontrolnych, a za przekształcenie ich w kształty podstawowe (trójkąty, odcinki) odpowiadają shadery powłoki (ang. *hull*) i dziedziny (ang. *domain*). Pierwszym krokiem potoku renderowania będzie jednak jak zwykle shader wierzchołków, gdzie każdy punkt kontrolny może być wstępnie przetworzony.

3.2 Shader powłoki

Punkty kontrolne trafiają następnie do shadera powłoki (linia 29). W nim każdy punkt kontrolny może być dodatkowo przetworzony korzystając z informacji o całym płatku. W szczególności liczba punktów kontrolnych na wejściu może być różna niż na wyjściu.

```
1  #define INPUT_PATCH_SIZE 3
2  #define OUTPUT_PATCH_SIZE 3
3  struct HSInput
4  {
5      float4 pos : POSITION;
6  };
7  struct HSPatchOutput
8  {
9      float edges[3] : SV_TessFactor;
10     float inside : SV_InsideTessFactor;
11 };
12 HSPatchOutput HS_Patch(
13     InputPatch<HSInput, INPUT_PATCH_SIZE> ip)
14 {
15     HSPatchOutput o;
16     o.edges[0] = o.edges[1] = o.edges[2] = 8.0f;
17     o.inside = 8.0f;
18     return o;
```

```
19 }
20 struct DSControlPoint
21 {
22     float4 pos : POSITION;
23 };
24 [domain("tri")]
25 [partitioning("integer")]
26 [outputtopology("triangle_cw")]
27 [outputcontrolpoints(OUTPUT_PATCH_SIZE)]
28 [patchconstantfunc("HS_Patch")]
29 DSControlPoint main(
30     InputPatch<HSInput, INPUT_PATCH_SIZE> ip,
31     uint i : SV_OutputControlPointID)
32 {
33     DSControlPoint o;
34     o.pos = ip[i].pos;
35     return o;
36 }
```

W powyższym przykładzie pierwszy parametr funkcji `main` (linia 30) zawiera listę punktów kontrolnych (typ `InputPatch<T, N>`, gdzie `T` to typ punktu kontrolnego zawierający atrybuty zwrócone z shadera wierzchołków, a `N` to liczba wejściowych punktów kontrolnych w płatku). W przykładzie każdy punkt kontrolny ma tylko przez pozycję, a płatek opisany jest przez 3 punkty kontrolne.

Drugi parametr `i` (o semantyce `SV_OutputControlPointID`) jest indeks wyjściowego punktu kontrolnego, na rzecz którego wywołany został shader powłoki. Na podstawie tych parametrów shader zwrócić powinien atrybuty `i`-tego wyjściowego punktu kontrolnego potrzebne shaderowi dziedziny.

Funkcja `main` opisana jest ponadto serią atrybutów określających parametry teselacji. Atrybut `domain` (linia 24) określa dziedzinę podziału. W zależności od tego jaki kształt chcemy dzielić mamy do wyboru:

- `"isoline"` — podział serii odcinków
- `"tri"` — podział trójkąta
- `"quad"` — podział kwadratu

Atrybut `partitioning` (linia 25) określa na ile fragmentów podzielona zostanie dziedzina w zależności od współczynników podziału (o których za chwilę). Współczynniki podziału będą miały typ `float` natomiast siłą rzeczy liczba fragmentów musi być całkowita. W zależności od wartości atrybuty współczynniki zaokrąglane są w górę do:

- `"pow2"` — potęgi liczby 2.
- `"integer"` — liczby całkowitej

- `"fractional_even"` — liczby parzystej
- `"fractional_odd"` — liczby nieparzystej

W przypadku dwóch ostatnich dwóch opcji dwa fragmenty będą pomniejszone względem reszty proporcjonalnie do różnicy pomiędzy wartością zaokrągloną a oryginalnym współczynnikiem podziału.

Trzeci atrybut `outputtopology` (linia 26) definiuje na jakie podstawowe kształty podzielona zostanie dziedzina:

- `"point"` —tylko punkty w wierzchołkach
- `"line"` — odcinki (dozwolony tylko dla dziedziny `"isoline"`)
- `"triangle_cw"` — trójkąty o orientacji zgodnej z ruchem wskazówek zegara (niedozwolony dla dziedziny `"isoline"`)
- `"triangle_ccw"` — trójkąty o orientacji przeciwnej z ruchem wskazówek zegara (niedozwolony dla dziedziny `"isoline"`)

Czwarty atrybut `outputcontrolpoints` (linia 27) definiuje ile punktów kontrolnych będzie na wyjściu shadera powłoki. Parametr `i` przyjmować będzie więc wartości od 0 do ilości podanej w tym atrybucie.

Ostatni atrybut `patchconstantfunc` (linia 28) określa nazwę funkcji stałych płatka. W przykładzie jest to funkcja `HS_Patch`, której definicja zaczyna się w linii 12. W przeciwieństwie do funkcji `main`, funkcja stałych wywoływana jest tylko raz dla całego płatka.

Parametrem tej funkcji jest lista wejściowych punktów kontrolnych (ta sama, co w funkcji `main`). Na wyjściu zwrócić musimy współczynniki podziału (jak widać w linii 7). W zależności od dziedziny muszą być to:

- dla `"isoline"`
  - tablica dwóch elementów typu `float` o semantyce `SV_TessFactor`. Pierwszy element określa na ile fragmentów podzielony zostanie każdy odcinek, drugi — ile odcinków powstanie;
- dla `"tri"`
  - tablica trzech elementów typu `float` o semantyce `SV_TessFactor`. Każdy element określa ilość wzdłuż jednej krawędzi trójkąta,
  - element typu `float` o semantyce `SV_InsideTessFactor` określający ilość podziałów wnętrza trójkąta (tj. ile fragmentów przetniemy idąc od jednej krawędzi, przez środek trójkąta, do innej krawędzi);
- dla `"quad"`
  - tablica czterech `float`-ów o semantyce `SV_TessFactor`. Każdy element określa ilość wzdłuż jednej krawędzi kwadratu,

– tablica dwóch `float`-ów o semantyce `SV_InsideTessFactor` określający ilość podziałów wnętrza kwadratu wzdłuż osi X i Y.

Nic oczywiście nie stoi na przeszkodzie, aby do struktury wyjściowej funkcji dodać dodatkowe pola na własne wartości stałe dla całego płatka, które przydać się mogą w shaderze dziedziny.

Nie zostało to zilustrowane w tym przykładzie ale w shaderze powłoki można korzystać również z buforów stałych, tekstur itp., w sposób podobny jak w shaderze wierzchołków. Więcej informacji o tworzeniu shaderów powłoki znaleźć można tutaj.

3.3 Shader dziedziny

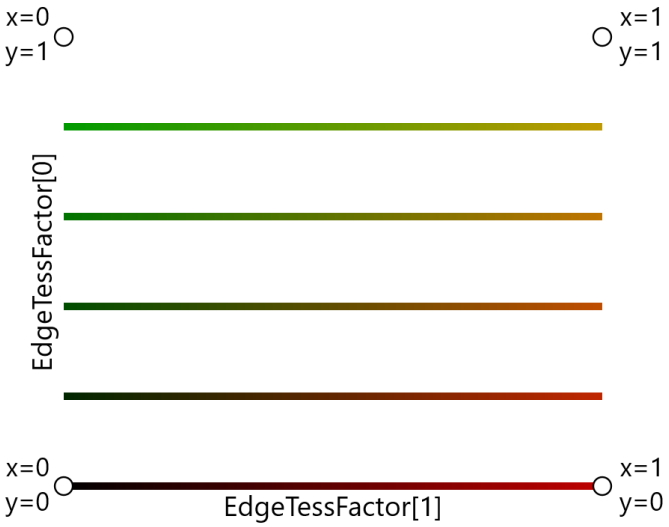
Wykonujący się automatycznie krok teselacji wyznacza pewne punkty wewnątrz wybranej dziedziny (wierzchołki trójkątów, końce odcinków, itp.). Dla każdego z nich wywoływany jest shader dziedziny, który powinien znaleźć wszystkie atrybuty tych wierzchołków. W naszym przykładzie sprowadzi się on do znalezienia pozycji punktów na ekranie, ale w bardziej skomplikowanych przypadkach wyznaczyć tu można inne atrybuty.

```
#define OUTPUT_PATCH_SIZE 3
cbuffer cbProj : register(b0)
{
    matrix projMatrix;
};
struct HSPatchOutput
{
    float edges[3] : SV_TessFactor;
    float inside : SV_InsideTessFactor;
};
struct DSControlPoint
{
    float4 pos : POSITION;
};
struct PSInput
{
    float4 pos : SV_POSITION;
};
[domain("tri")]
PSInput main(
    HSPatchOutput factors,
    float3 uvw : SV_DomainLocation,
    const OutputPatch<DSControlPoint, OUTPUT_PATCH_SIZE> i)
{
    PSInput o;
    float4 pos = i[0].pos * uvw.x +
                 i[1].pos * uvw.y +
                 i[2].pos*uvw.z;
```

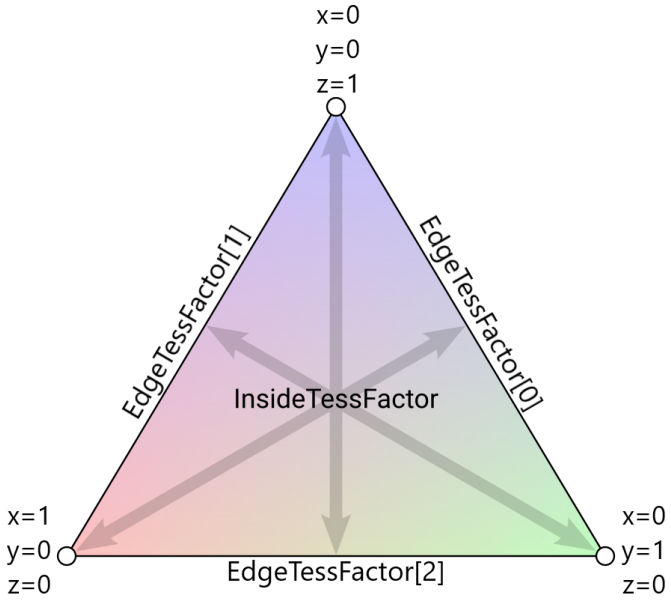
```
29      o.pos = mul(projMatrix, pos);
30      return o;
31  }
```

Parametrami funkcji `main` (20) shadera dziedziny mogą być m.in.:

- lista wyjściowych punktów kontrolnych (linia 23) — stała lista punktów kontrolnych, wyznaczonych w shaderze powłoki
- Atrybuty stałe dla całego płatka wyznaczone przez funkcję stałych płatka shadera powłoki (w przykładzie, linia 21), w tym współczynniki podziału.
- Współrzędne dziedziny (semantyka `SV_DomainLocation`, w przykładzie linia 22), zależnie od jej typu:
  - dla `"isoline"` — typu `float2`

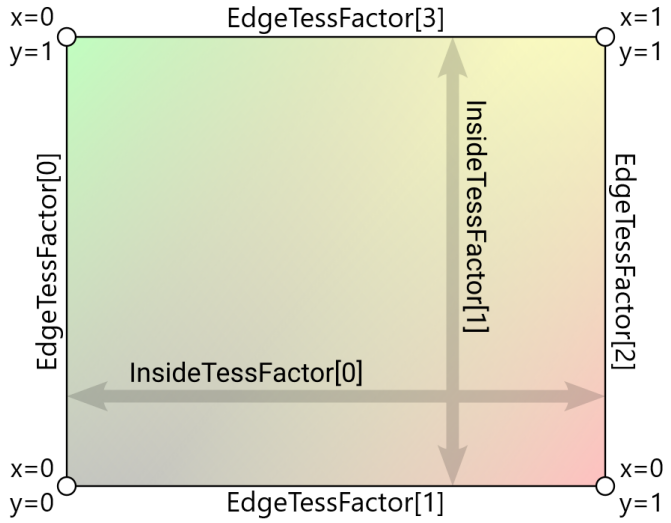


    Współrzędne wskazują na pewien punkt w przedziale $[0,1]\times[0,1]$ jednej z linii ułożonych są wzdłuż osi X. Poszczególne linie oddalone są od siebie o pewien odstęp wzdłuż osi Y. Warto zauważyć, że nie ma linii $y = 1$.
  - dla `"tri"` — typu `float3`



Współrzędne barycentryczne ($x^2 + y^2 + z^2 = 1$) wskazujące na punkt wewnątrz trójkąta.

– dla `"isoline"` — typu `float2`



Współrzędne wskazują na pewien punkt w przedziale $[0, 1] \times [0, 1]$.

Typ dziedziny definiuje atrybut `domain` (linia 19) funkcji `main`. Shader dziedziny można niejako traktować shader wierzchołków dla wierzchołków prostych kształtów geometrycznych (punktów, odcinków, trójkątów) powstałych w wyniku teselacji.

Warto na koniec wspomnieć, że wynik shadera dziedziny nie musi być przekazany bezpośrednio do rasteryzacji. Jeżeli chcemy poszczególne proste

kształty (trójkąty, odcinki, punkty) biorąc pod uwagę ich wszystkie wierzchołki, możemy do potoku renderowania dołączyć jeszcze shader geometrii. Umożliwiłby nam on m.in. dokonanie kolejnych podziałów, zmianę typu kształtu (np. wygenerowanie bilboardu dla punktu lub odcinka), czy wyznaczenie wartości których nie da się policzyć na podstawie pojedynczego wierzchołka (np. wyznaczenie wektora normalnego do płaszczyzny trójkąta).

3.4 Użycie shaderów teselacji

Tak jak w przypadku innych typów, obiekty shaderów powłoki i dziedziny można stworzyć na podstawie skompilowanego kodu wykorzystując funkcje `CreateHullShader` i `CreateDomainShader` interfejsu `ID3D11Device`.

Stworzone tak obiekty dodać można do potoku renderowania korzystając z funkcji `HSSetShader` i `DSSetShader` interfejsu `ID3D11DeviceContext`:

```
context->HSSetShader(hs, nullptr, 0);
context->DSSetShader(ds, nullptr, 0);
```

Oba etapy mają też swój zestaw funkcji do dołączania używanych buforów stałych, tekstur i samplerów:

- `HSSetConstantBuffers`, `HSSetShaderResources`, `HSSetSamplers`
- `DSSetConstantBuffers`, `DSSetShaderResources`, `DSSetSamplers`

Przed wywołaniem funkcji rysującej (np. `Draw`, czy `DrawIndexed`) należy wywołać jeszcze funkcję `IASetPrimitiveTopology` przekazując wartość odpowiadającą liczbie wejściowych punktów kontrolnych shadera powłoki. W naszym przykładzie korzystaliśmy z 3 punktów kontrolnych — ilość tą widać w typie pierwszego parametru (linia 30) funkcji `main` shadera powłoki. Odpowiadające tej ilości wywołanie wyglądać powinno:

```
context->IASetPrimitiveTopology(
    D3D11_PRIMITIVE_TOPOLOGY_3_CONTROL_POINT_PATCHLIST);
```