

**WYDZIAŁ  
ELEKTROTECHNIKI  
I INFORMATYKI  
POLITECHNIKI RZESZOWSKIEJ**

**Katedra Elektrotechniki i Podstaw Informatyki**

**Jakub Biel**

Wizualizacja i zastosowanie szumu w grafice komputerowej

**Praca dyplomowa inżynierska**

Opiekun pracy:  
dr inż. Grzegorz Drałus

Rzeszów, 2016



# Spis treści

<b>1. Wprowadzenie</b>	<b>6</b>
1.1. Motywacja	6
1.2. Współczesne aplikacje szumu	6
1.3. Cel i Zakres Pracy	9
<b>2. Wprowadzenie Teoretyczne</b>	<b>10</b>
2.1. Szumy	10
2.1.1. Value Noise	10
2.1.2. Szum Perlina	12
2.1.3. Szum Worley'a	13
2.2. Szumy Fraktalne	13
2.3. Szumy Multifraktalne	15
<b>3. Budowa Aplikacji</b>	<b>17</b>
3.1. Zastosowane narzędzia oraz technologie	17
3.2. Implementacja Szumów	17
3.2.1. Value Noise	17
3.2.2. Szum Perlin'a	18
3.2.3. Szum Worley'a	20
3.2.4. Szum Fraktalny	22
3.3. Architektura Aplikacji	23
3.3.1. Generacja Terenu	23
3.3.2. Generacja Tekstur	23
<b>4. Obsługa Aplikacji</b>	<b>24</b>
4.1. Modyfikacja parametrów generacyjnych	25
4.2. Eksport geometrii siatki terenu oraz tekstur	27
<b>5. Metody tworzenia treści z wykorzystaniem szumów fraktalnych</b>	<b>29</b>
5.1. Wykorzystanie podstawy sinusoidalnej	29
5.2. Ridged Perlin Noise	31
5.3. Wariacje szumu Worley'a	33
5.4. Zastosowanie funkcji blend	35
<b>6. Podsumowanie i wnioski końcowe</b>	<b>38</b>
<b>Załączniki</b>	<b>40</b>





# **1. Wprowadzenie**

## **1.1. Motywacja**

Wiarygodność wirtualnego środowiska zależy od jakości odwzorowania rzeczywistego elementu świata. Utworzenie autentycznego środowiska wymaga niezwykłej ilość pracy do tworzenia tekstur, modelowania świata itp. Dodatkowym problemem jest rozmiar tworzonych elementów na dysku, który łatwo może przekroczyć dopuszczalne miejsce przyjmując realizm jako priorytet.

Jednym z rozwiązań tego problemu są metody proceduralne, które w pewien sposób aproksymują rzeczywiste jakości świata. Pozwala to nie tylko na zmniejszenie wymaganego miejsca do pewnej transformacji matematycznej, ale też na rozszerzenie metody do obejmowania generacji całego świata, istotnie zmniejszając wymaganą pracę.

Metody proceduralne pozwalają na unikalność generowanych treści, poprzez umożliwienie generacji nieskończonej ilość nieskończonych wariacji, tworzonych za pomocą ustalonych reguł.

Jedną z najbardziej znaczących metod pozwalających na generacje szerokiej gamy elementów jest szum fraktalny, a w szczególność szum perlina, dlatego będzie on głównym tematem tej pracy.

## **1.2. Współczesne aplikacje szumu**

Szum ma szerokie zastosowania w dziedzinie grafiki komputerowej. Jest używany do m.in. generacji i syntezy tekstur, generacji terenu oraz bardziej organicznego rozmieszczania obiektów.

Minecraft - jedna z najpopularniejszych współcześnie gier, która używa proceduralnej generacji do utworzenia świata[2]. Pozwala na eksploracje niemal nieskończonego świata, gdzie każda strefa klimatyczna ma charakterystyczne dla siebie elementy.



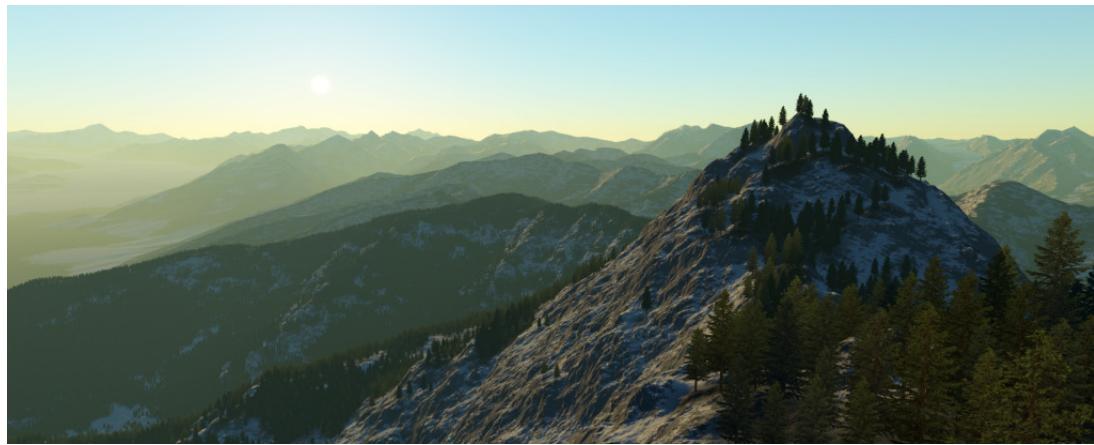
Rys. 1.1. Minecraft

Terragen - Generator scenerii dla artystów, który używa szumów do generacji świata (terenu i tekstury).



Rys. 1.2. Terragen

Outerra - Używa danych wysokościowych terenu do budowania siatki powierzchni ziemi. By dodać detale do map źródłowych o niskiej rozdzielczości, używane są algorytmy fraktalne(m.in. szумy) produkujące detale w skali milimetra.



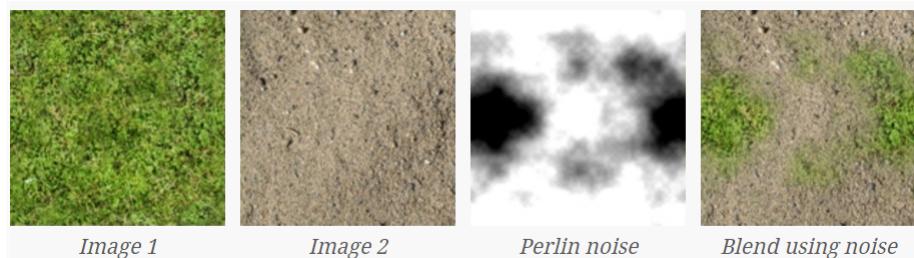
Rys. 1.3. Outerra

No Man's Sky - Gra pozwalająca na eksplorację proceduralnie wygenerowanego wszechświata, w którym można odnaleźć 18 kwintylionów unikalnych planet.



Rys. 1.4. No Man's Sky

Oprócz powyższych oczywistych zastosowań, w wielu aplikacjach graficznych szum używany jest m. in. do blendowania (Zestawienia) 2 tekstur.



Rys. 1.5. Ilustracja zestawienia 2 tekstur przy pomocy szumu perlina.[6]

### 1.3. Cel i Zakres Pracy

Celem pracy jest przedstawienie sposobów generacji tekstur, terenu i transformacji z wykorzystaniem szumów fraktalnych. Będzie przedstawiony także sposób implementacji proceduralnie generowanego ‘w locie’ terenu, który może być użyty w grach lub aplikacjach o podobnych wymaganiach.

Praca obejmuje opracowanie teoretycznych zagadnień, szczegółową implementację algorytmów. Praktyczne zastosowanie poznanych metod do wygenerowania elementów o odpowiednich jakościach oraz opracowanie aplikacji ułatwiającej generację tych elementów.

## 2. Wprowadzenie Teoretyczne

### 2.1. Szumy

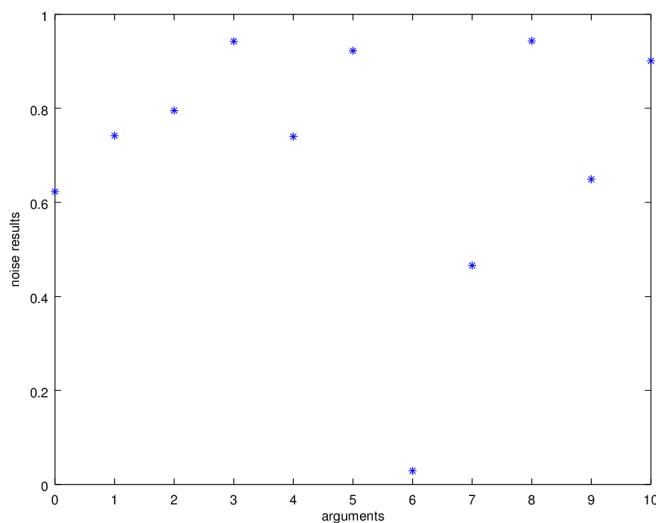
Szum używany w grafice jest to funkcja matematyczna przyjmująca  $n$ -wymiarowy wektor i zwracająca wartość dla danego argumentu. W zależności od parametrów funkcji i wymiarowości funkcja ta może reprezentować szereg elementów.

Jednowymiarowa funkcja może reprezentować falę dźwiękową, czy ukształtowanie terenu widzianego z jednej ze stron. Dwuwymiarowy szum może być heightmap'ą terenu tzn. ukształtowaniem terenu, gdzie wyższa wartość reprezentuje wyższa pozycje wierzchołka siatki geometrii. Tekstura może być 2-wymiarowym szumem, gdzie kolor przypisany jest do odpowiedniej wartości szumu. 3-wymiarowy szum tworzy nieregularne 3-wymiarowe kształty, które mogą być użyte np. jako formacje komór jaskiń czy volumetryczne chmury.

Głównym tematem pracy jest generacje terenu oraz tekstur, dlatego szczególnie opisywany jest 2 wymiarowy szum.

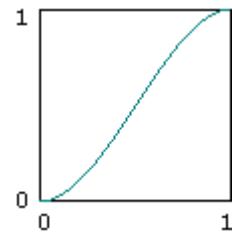
#### 2.1.1. Value Noise

Value Noise to najprostszy rodzaj szumu. Opiera się on na przyporządkowaniu do punktów o koordynatach całkowitych wartości w funkcji pozycji. Wartość każdego próbkowanego punktu będzie determinowana poprzez interpolacje sąsiednich punktów[7].

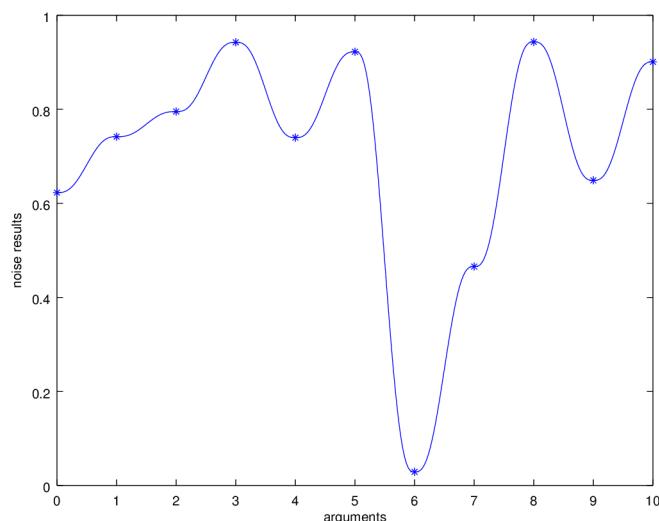


Rys. 2.6. Ilustracja przyporządkowania punktów dla jednowymiarowego szumu.

Można zastosować różna rodzaje interpolacji dla określania wartości próbkowanego punktu (np. cosinosową czy szesienną), lecz ze względu na szybkość używana jest interpolacja liniowa z wykorzystaniem funkcji wygładzającej dla interpolantu  $t$ .



Rys. 2.7. Funkcja wygładzająca (ease curve) ( $6t^5 - 15t^4 + 10t^3$ )



Rys. 2.8. Wartość szumu obliczona za pomocą interpolacji liniowej z funkcją wygładzającą interpolant  $t$

W przypadku szumu 2-wymiarowego wartość próbkowanego punktu jest określana na podstawie sąsiednich 4 punktów z użyciem interpolacji 2 liniowej (3-krotnie zastosowanie interpolacji liniowej).

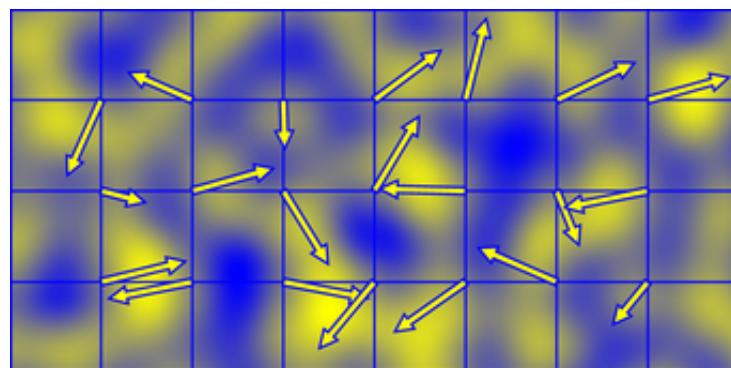


Rys. 2.9. Wartość szumu 2-wymiarowego

Value noise szumu tworzy poziome i pionowe(kapsułkowe) wzory w 2-wymiarach, dlatego nie jest najbardziej przydatnym szumem, aczkolwiek ma specyficzne zastosowania.

### 2.1.2. Szum Perlina

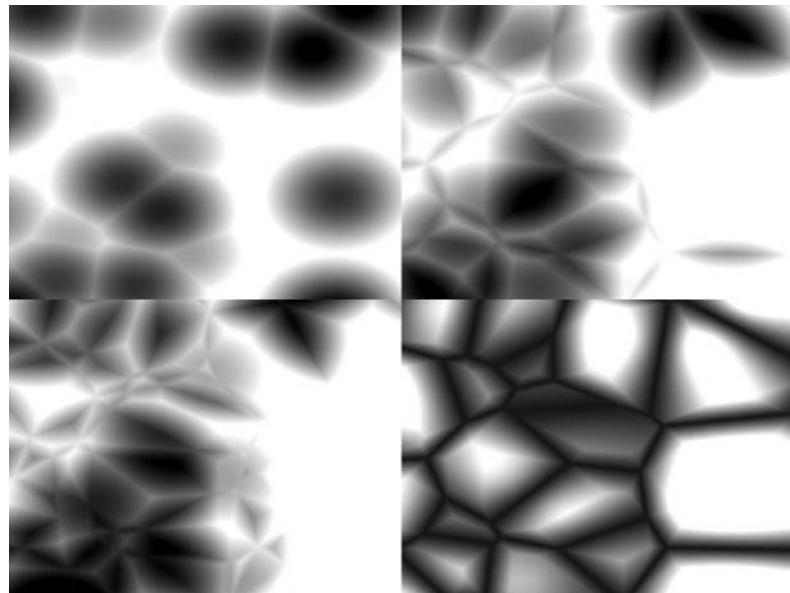
Szum Perlina(Szum gradientowy) jest bardzo podobny w definicji do Value Noise, różnicą jest wartość przypisywana do koordynatów całkowitych. W przeciwieństwie do Value Noise zamiast zwykłej wartości przypisany jest wektor, za pomocą którego określana jest wartość próbkowanego punktu[8]. Zastosowanie gradientów pozwala na pozbycie się artefaktów występujących w Value Noise.



Rys. 2.10. Dwuwymiarowy szum perlina z nałożonymi wektorami punktów o koordynatach całkowitych.

### 2.1.3. Szum Worley'a

Worley Noise - Szum Komórkowy. Próbkowany punkt ma wartość zależną od odległości od najbliższych punktów. Punkty są losowo rozmieszczone z zapewnieniem pewnej gęstości. Istnieje wiele podrodzajów szumu Worley'a z powodu wielu metryk odległość jakie można zastosować oraz wyboru punktów używanych do determinacji odległości. Nadaje się on idealnie do tworzenia wzorów o strukturze nieregularnych kafelków.

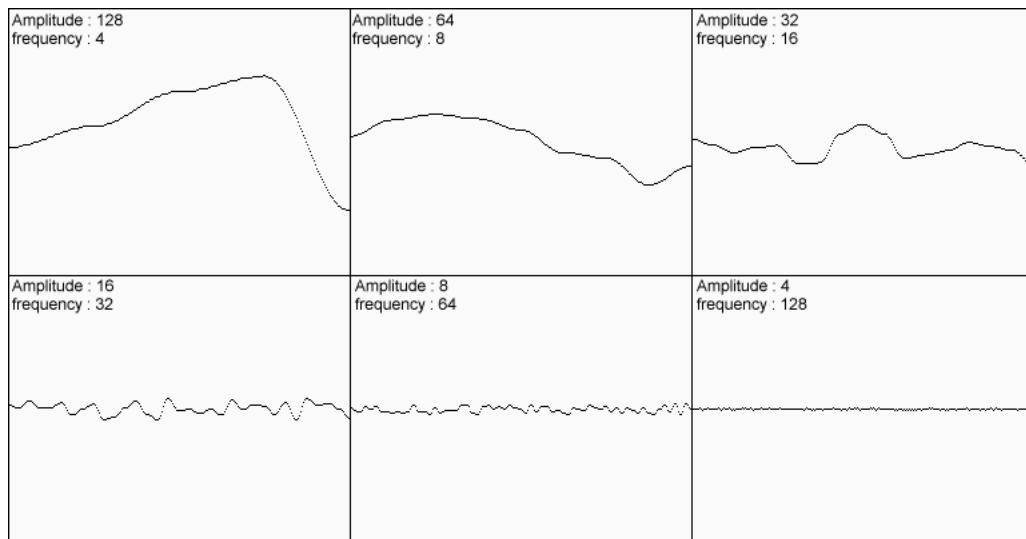


Rys. 2.11. Szum Worley'a - klasyfikatory  $f_1, f_2, f_3, f_2 - f_1$ , metryka euklidesowa.

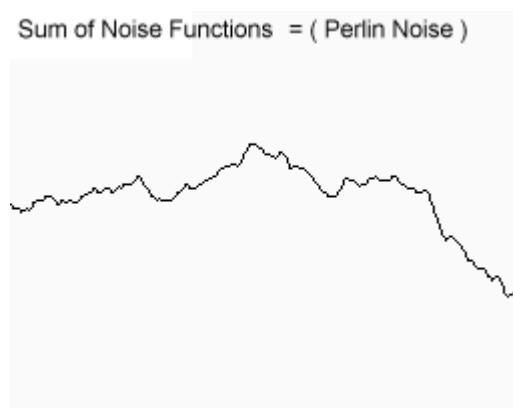
Oznaczenia  $f_1, f_2, f_3$  określają wartość szumu, gdzie  $f_1$  to odległość od najbliższego punktu,  $f_2$  od drugiego najbliższego itd. W przypadku  $f_2 - f_1$  wartością szumu jest różnica odległości pomiędzy dwoma najbliższymi punktami.

## 2.2. Szумy Fraktalne

Szum fraktalny (nazywany także z ang. Fractional/Fractal Brownian Motion) to nałożone na siebie wartości szumów o coraz wyższych częstotliwościach z odpowiednimi wagami. W efekcie każda kolejna wartość szumu jest coraz mniej istotna, efektywnie dodając coraz mniejsze szczegóły.



Rys. 2.12. Ilustracja 6 oktaw szumu perlina z odpowiednio zmniejszoną amplitudą



Rys. 2.13. Kombinacja powyższych oktaw z rys. 2.12

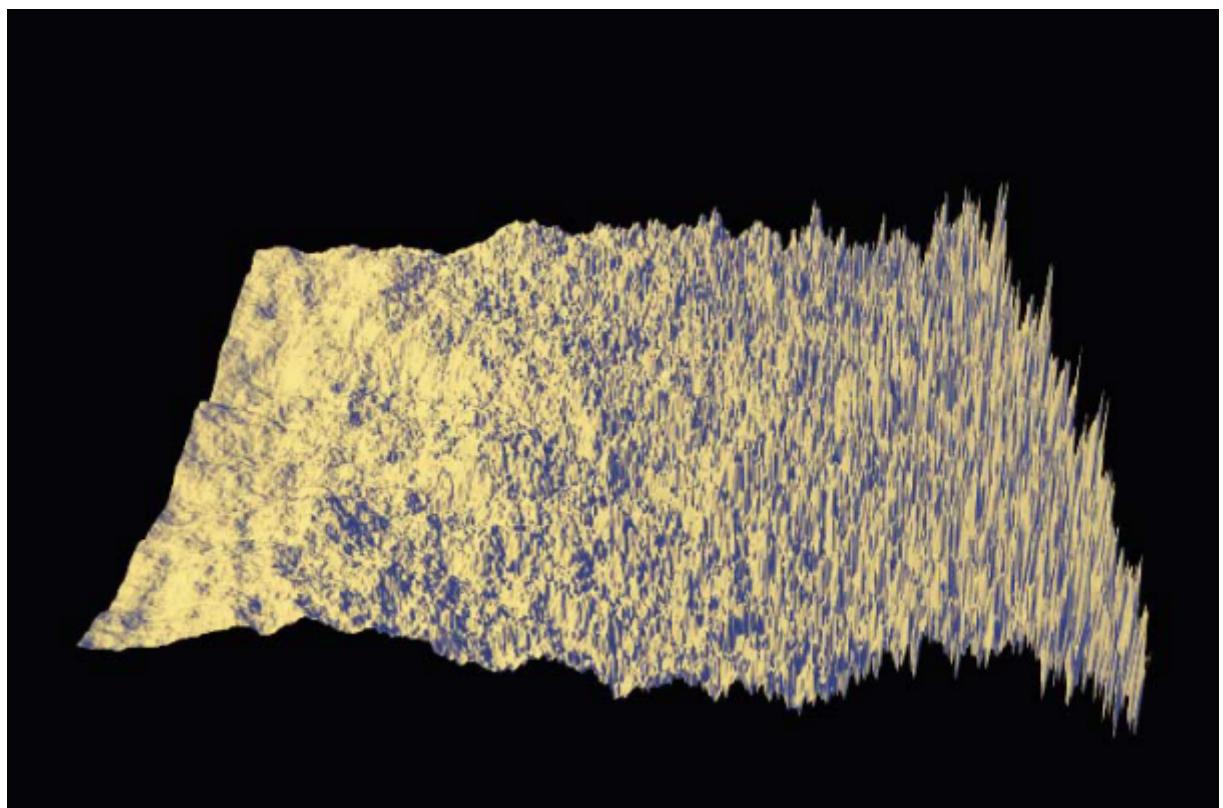
Parametry szumów fraktałnych:

- Frequency - Częstotliwość bazowa.
- Liczba Oktaw - Ilość warstw nałożonych na siebie szumów składających się na wartość końcową.
- Lacunarity - to mnożnik częstotliwości dla każdej kolejnej oktawy. Przy Częstotliwości bazowej 1.0 i Lacunarity 2.0 kolejne mapy będą miały częstotliwości o wartości: 1.0, 2.0, 4.0, 8.0 itd.
- Persistence - Waga każdej kolejnej oktawy tzn. Przy persistence 0.5 kolejne wagi będą następujące: 1.0, 0.5, 0.25, 0.125 itd.

### 2.3. Szumy Multifraktalne

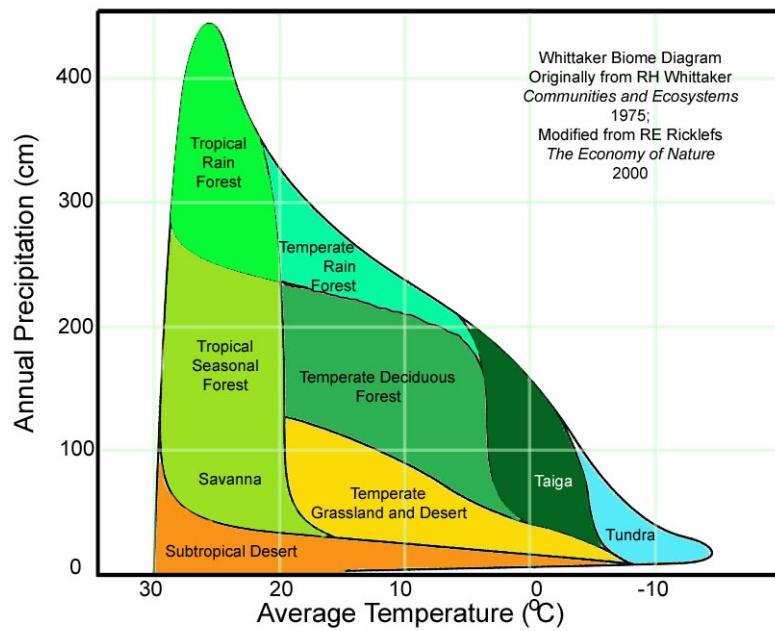
Szumy fraktalne pozwalają na generacje wartości o określonej wariacji, często jednak jakości samego szumu fraktalnego są nieciekawe, w przypadku generacji height-map'y wszystkie góry mają szczyty na bardzo podobnej wysokości.

Poprzez odpowiednie złożenie wielu szumów fraktalnych istnieje możliwość zróżnicowania terenu na wyższym poziomie. Najprostszym przykładem jest przemnożenie fraktalnego szumu perlina przez szum perlina o dużo niższej częstotliwości w efekcie tworząc tereny górzyste oraz równiny, a więc teren ciekawszy niż każdy ze szumów składowych.



Rys. 2.14. Teren Multifraktalny wartość szumu o niższej częstotliwości maleje z prawej do lewej

Jeżeli zastosujemy model klimatyczny Whittaker'a[4] możemy użyć dwóch szumów fraktalnych do reprezentacji temperatury oraz opadów. Następnie w odpowiednich przedziałach powyższych szumów użyć szumu fraktalnego reprezentującego ukształtowanie terenu danego klimatu.



Rys. 2.15. Diagram Whittakera gdzie oś pozioma to średnia temperatura, oś pionowa to roczne opady

Poprzez odpowiednią kombinację podstawowych funkcji takich jak sin, cos, mod, abs oraz szumów fraktalnych można wygenerować niemal każdy teren i każdą strukturę powierzchni.

### 3. Budowa Aplikacji

Celem utworzonej aplikacji jest ułatwienie generacji treści opartej na szumach.

#### 3.1. Zastosowane narzędzia oraz technologie

Aplikacja została wykonana z użyciem języka C++ oraz biblioteki graficznej OpenGL.

Jako biblioteki pomocnicze użyto:

- 1) FreeGlut - do tworzenia okna i obsługi zdarzeń.
- 2) Glew - do określania zgodności funkcji api OpenGL'a.
- 3) AntTweakBar - do tworzenia i obsługi interfejsu użytkownika.

Jako edytor tekstu został użyty emacs, natomiast kompilator i linker użyty do budowy aplikacji pochodzi z pakietu Visual Studio 2015. Jako debugger zostało użyte środowisko programistyczne Visual Studio 2015.

#### 3.2. Implementacja Szumów

W implementacji szumów wykorzystywana jest tablica hash przechowujące wartości w zakresie od 0 do 255. Wartości tej tablicy są te same co w kodzie ulepszzonego szumu Ken'a Perlin'a[3]. Użycie tej tablica pozwala na zapewnienie odpowiedniej losowości.

##### 3.2.1. Value Noise

Implementacja 2-wymiarowego Value Noise.

```
1 real32
2 Noise::value(Vec2f point, real32 frequency)
3 {
4     point *= frequency;
5     int32 ix0 = (int32)floor(point.x);
6     int32 iy0 = (int32)floor(point.y);
7     real32 tx = point.x - ix0;
8     real32 ty = point.y - iy0;
9     ix0 &= hashMask;
10    iy0 &= hashMask;
11    int32 ix1 = ix0 + 1;
```

```

12     int32 iy1 = iy0 + 1;
13     iy1 &= hashMask;
14     int32 h0 = hash[ix0];
15     int32 h1 = hash[ix1];
16     int32 h00 = hash[h0 + iy0];
17     int32 h10 = hash[h1 + iy0];
18     int32 h01 = hash[h0 + iy1];
19     int32 h11 = hash[h1 + iy1];
20     tx = smooth(tx);
21     ty = smooth(ty);
22     real32 upX = lerp((float)h00, (float)h10, tx);
23     real32 downX = lerp((float)h01, (float)h11, tx);
24     real32 middle = lerp(upX, downX, ty);
25     return middle / 255.0f;
26 }
```

Najpierw determinowane są współrzędne lewego górnego rógów kwadratu mieszczącego próbowany punkt za pomocą funkcji *floor*.

Następnie obliczana jest delta od tego rogu - *tx,ty*. Zapewniane jest także by wartości były mniejsze od wielkości tablicy hash poprzez operacje iloczynu bitowego z maską. Obliczane są także wartości pozostałych potrzebnych koordynatów całkowitych.

Następnie odczytywane są wartości w rogach kwadratu z tablicy hash. Kolejno stosowana jest funkcja wygładzająca by określić słuszniejsze wartości interpolantów. Na zakończenie zastosowana jest dwuliniowa interpolacja i normalizacja wyniku.

Gdzie: *lerp* - to funkcja dokonująca interpolacji liniowej.

### 3.2.2. Szum Perlin'a

Implementacja 2-wymiarowego Szumu Perlina.

```

1 real32
2 Noise::perlin(Vec2f point, real32 frequency)
3 {
4     point *= frequency;
5     int32 ix0 = (int)floor(point.x);
6     int32 iy0 = (int)floor(point.y);
7     real32 tx0 = point.x - ix0;
8     real32 tx1 = tx0 - 1.0f;
9     real32 ty0 = point.y - iy0;
```

```

10    real32 ty1 = ty0 - 1.0f;
11    ix0 &= hashMask;
12    int32 ix1 = ix0 + 1;
13    iy0 &= hashMask;
14    int32 iy1 = iy0 + 1;
15    iy1 &= hashMask;
16    int32 h0 = hash[ix0];
17    int32 h1 = hash[ix1];
18    Vec2f g00 = gradients2D[hash[h0 + iy0] & gradients2DMask];
19    Vec2f g10 = gradients2D[hash[h1 + iy0] & gradients2DMask];
20    Vec2f g01 = gradients2D[hash[h0 + iy1] & gradients2DMask];
21    Vec2f g11 = gradients2D[hash[h1 + iy1] & gradients2DMask];
22    real32 v00 = Vec2f::dotProduct(g00, tx0, ty0);
23    real32 v10 = Vec2f::dotProduct(g10, tx1, ty0);
24    real32 v01 = Vec2f::dotProduct(g01, tx0, ty1);
25    real32 v11 = Vec2f::dotProduct(g11, tx1, ty1);
26    real32 tx = smooth(tx0);
27    real32 ty = smooth(ty0);
28    real32 upX = lerp(v00, v10, tx);
29    real32 downX = lerp(v01, v11, tx);
30    return lerp(upX, downX, ty) * sqr2;
31 }

```

Podobnie jak w kodzie Value Noise kalkulujemy koordynaty rogów 'kwadratu' mieszczącego próbkowany punkt. Po określeniu wektorów dla każdego z rogów następuje obliczanie końcowej wartości, poprzez obliczenie iloczynu skalarnego wektora z odpowiednimi deltami  $tx$  oraz  $ty$ . Następnie wygładza się współczynnik  $t$  w każdym wymiarze oraz interpoluje dwuliniowo by otrzymać wynik.

Najwyższą wartość może mieć środek 'kwadratu' kiedy wektory rogów są skierowane ku środkowi. Tą maksymalną wartością jest  $\sqrt{1/2}$  dlatego by znormalizować wartość należy ją pomnożyć przez  $\sqrt{2}$  [5].

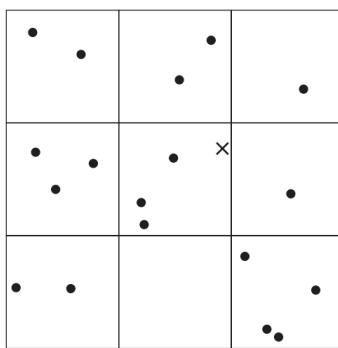
Gdzie:

- `gradients2D` - To statyczna tablica wektorów.
- `dotProduct` - To metoda obliczająca iloczyn skalarny o prototypie:  
 $real32 dotProduct(v1, v2x, v2y)$ .
- `lerp` - To funkcja dokonująca interpolacji liniowej.

### 3.2.3. Szum Worley'a

By określić wartość szumu Worley'a potrzeba odnaleźć odległość dwóch najbliższych punktów wedle określonej metryki odległości. By można było to robić w sposób deterministyczny dokonuje się segmentacji przestrzeni wedle koordynatów całkowitych, a następnie na podstawie hash'a pozycji określana jest ilość i pozycje punktów w danym segmencie [1].

By znacznie uprościć algorytm założono, że w segmencie gdzie znajduje się próbowany punkt oraz 8 sąsiednich znajdują się przynajmniej 2 punkty. By zminimalizować ryzyko braku 2 punktów do określenia ich ilości dla chunk'a zastosowano rozkład Poissona o współczynniku lambda 1.5.



Rys. 3.16. Ilustracja segmentacji przestrzeni.

---

```
1 real32
2 Noise::worley(Vec2f point, real32 frequency, WORLEY_TYPE worleyType, ←
    DISTANCE_TYPE distanceType)
3 {
4     point *= frequency;
5     Vec2i currentChunkPosition = Vec2i(floor(point.x), floor(point.y));
6     std::list<real32> distanceList;
7     for (auto it = chunksToCheck.begin(); it != chunksToCheck.end(); it++)
8     {
9         Vec2i& chunkOffset = *it;
10        std::vector<Vec2f> cubeMarksOffset = worleyGetPoints(←
11            currentChunkPosition + chunkOffset);
12        addDistances(point, cubeMarksOffset, distanceList, distanceType);
13    }
14    distanceList.sort();
```

```

14     real32 f1 = 0;
15     real32 f2 = 1.0f;
16     if(distanceList.size() > 1)
17     {
18         f1 = *distanceList.begin();
19         f2 = *(++distanceList.begin());
20     }
21     else
22     {
23         std::cout << "we";
24     }
25     real32 worleyResult;
26     switch (worleyType)
27     {
28         case WT_F2SUBF1: worleyResult = f2 - f1; break;
29         case WT_F1: worleyResult = f1; break;
30         case WT_F2: worleyResult = f2; break;
31         case WT_F2ADD1: worleyResult = f2 + f1; break;
32         case WT_F1MULF2: worleyResult = f1 * f2; break;
33         default: std::cout << "No such worley type \n";
34     }
35     return worleyResult;
36 }
```

Najpierw określana jest pozycja 'kwadratu', w którym znajde się próbkiowy punkt. Następnie do listy odległości od punktów w pętli dodawane są odległość z obecnego 'kwadratu' oraz 8 sąsiednich. Po posortowaniu pobierane są 2 najmniejsze wartości ( $f_1$  oraz  $f_2$ ) na podstawie, których określana jest wartość szumu.

### 3.2.4. Szum Fraktalny

Ogólny postać kodu obliczającego szum fraktalny:

```
1 real32
2 Noise::sum( const Vec2f& point, const NoiseParams& noiseParams ) {
3     real32 sum = 0;
4     real32 amplitude = 1.0f;
5     real32 range = 0;
6     real32 frequency = noiseParams.frequency;
7     for( int32 i = 0; i < noiseParams.octaves; i++ )
8     {
9         range += amplitude;
10        sum += noise( point, frequency ) * amplitude;
11        frequency *= noiseParams.lacunarity;
12        amplitude *= noiseParams.persistence;
13    }
14    return sum / range;
15 }
```

W pętli sumowana jest wartość szumu z odpowiednią amplitudą. Na zakończenie wartość jest normalizowana czyli dzielona przez sumę amplitud szumów składowych.

Gdzie noise - to dowolna funkcja obliczająca wartość szumu.

### **3.3. Architektura Aplikacji**

Istnieją dwa główne moduły aplikacji - Moduł generacji terenu oraz Moduł odpowiedzialny za generacje tekstur.

#### **3.3.1. Generacja Terenu**

By ułatwić generacje, teren jest podzielony na łatwiej zarządzalne fragmenty (Chunki). Główny algorytm działa w następujący sposób.

- 1) Wybierana jest ilość fragmentów(chunków) terenu które powinny być wygenerowane.
- 2) Jeżeli któryś z chunków nie jest wygenerowany lub nie generuje się obecnie to tworzony jest wątek, który generuje zlecony chunk.
- 3) Usuwane są nie wymagane chunki.

Na podstawie pozycji kamery determinowane są chunk'i które powinny być generowane. Zaimplementowany system generacji pozwala na tworzenie chunków w locie, jeżeli maszyna jest wystarczająca szybka, generacja działa w sposób niedostrzegalny, dając wrażenie eksploracji rzeczywistego terenu.

Chunk'om do generacji przekazywany jest 2-wymiarowy wektor indeksu pozycji oraz wyrażenie i definicje szumów znajdujących się w wyrażeniu (Rodzaj szumu, częstotliwość itd.).

Wierzchołki geometrii są kolorowane na podstawie wysokości i określonych kolorów ustawianych w panelu bocznym. Kolorowanie wierzchołku następuje w shaderze - do shadera przekazywana jest lista kolorów wraz z wartościami początkowymi. Kolory z panelu bocznego są współdzielone z modułem tekstur.

Po wygenerowaniu heightmapy dla danego Chunk'u następuje kalkulacja normalnych, by można było symulować oświetlenie, a następnie dane kopowane są do pamięci karty graficznej (W wątku głównym).

#### **3.3.2. Generacja Tekstur**

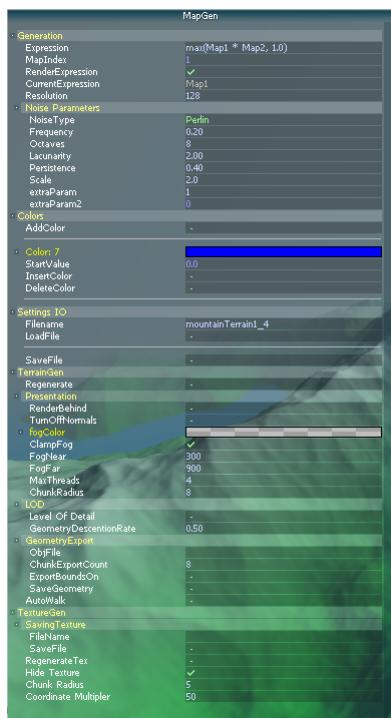
Działa w bardzo podobny sposób jak generacja terenu z tym, że zamiast wykorzystania wygenerowanej heightmapy do generacji terenu jest ona kolorowana i użyta jako tekstura.

Sposób w jaki kolorowana jest tekstura można modyfikować w panelu bocznym.

Regeneracja tekstury następuje po zmianie jednego z istotnych parametrów generacyjnych. Panel parametrów generacyjnych tekstur jest współdzielony z modułem generacji terenu.

## 4. Obsługa Aplikacji

Interfejs użytkownika został utworzony z wykorzystaniem biblioteki AntTweakBar.



Rys. 4.17. Panel boczny aplikacji.

Istnieje pięć głównych zakładek w Panelu Bocznym.

- 1) Generation - definiująca parametry generacyjne - wyrażenia, definicje szumów, rozdzielcość generacji itp.
- 2) Colors - opisująca kolory które są użyte przy renderowaniu tekstur i terenu.
- 3) Settings IO - jest to karta umożliwiająca zapis i odczyt definicji generacyjnych (Elementy z zakładek Generation i Colors).
- 4) TerrainGen - Opcje specyficzne dla generacji terenu np. promień generacji chun-ków, opcja eksportu geometrii do pliku.
- 5) TextureGen - Opcje specyficzne dla generacji tekstur.

## 4.1. Modyfikacja parametrów generacyjnych

Pola Zakładki Generation:

- 1) Expression - Wyrażenie określające generacje.
- 2) MapIndex - Indeks mapy której parametry można obecnie modyfikować w karcie Noise Parameters.
- 3) RenderExpression - Flaga określająca czy renderowane jest wyrażenie czy mapa z pola mapIndex.
- 4) CurrentExpression - Obecnie używane wyrażenie(Na wypadek wprowadzonego złego wyrażenia w polu expression).
- 5) Resolution - Rozdzielcość generowanego fragmentu terenu.

Najbardziej istotnym elementem charakteryzującym generacje jest wartość wyrażenia, które obecnie określa sposób generacji. Jest to pierwsze pole tekstowe w panelu Generation. Można w nim wykorzystać podstawowe funkcje takie jak: *min*, *max*, *sin*, *cos*, *abs*, *mod*, *floor*, *ceil*. Można także używać zmiennych takich jak *x* i *y* które przyjmują wartość pozycji generowanego wierzchołka.

Pole MapIndex opisuje indeks obecnej mapy którą można edytować w podkategorii NoiseParameters. Wartość szum fraktalnego można wykorzystać w wyrażeniu wykorzystując zmienną o nazwie Map[IndeksMapy]. Gdzie [IndeksMapy] to indeks wybranego szumu fraktalnego.

Istnieje także prosta funkcja blend która może być użyta do prostego łączenia dwóch szumów fraktalnych.

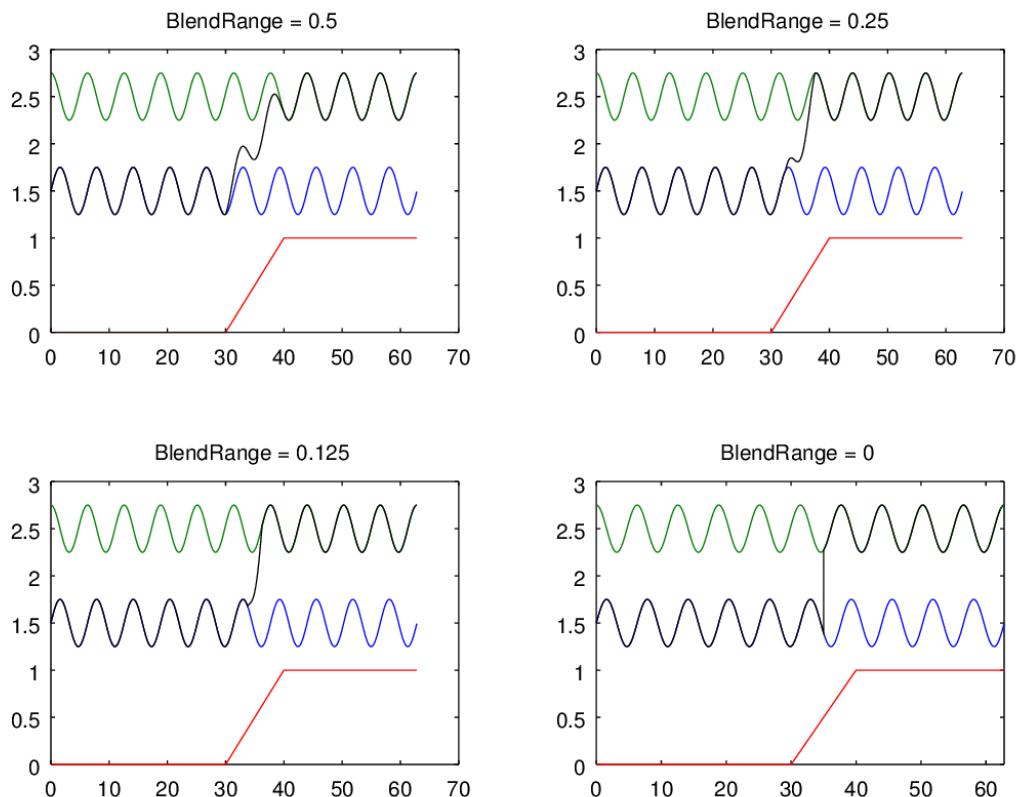
Kod funkcji blend:

```
1 real32
2 blendSmooth(real32 val1, real32 val2, real32 t, real32 blendRange)
3 {
4     real32 result = 0;
5     if(t > (0.5f - blendRange) && t < (0.5f + blendRange))
6     {
7         real32 resultT = ((t - (0.5f - blendRange)) / (blendRange * 2.0f));
8         result = val1 + (val2 - val1) * resultT;
9     }
10    else if(t < 0.5f) result = val1;
```

```

11     else result = val2;
12
13 }
```

Działa ona w następujący sposób.



Rys. 4.18. Ilustracja działania funkcji blend.(kolor niebieski - *val1*, kolor zielony - *val2*, kolor czerwony - interpolant *t*, kolor czarny - wynik funkcji)

Funkcja blendSmooth pozwala w prosty sposób połączyć dwa szумy fraktalne w sposób nie tworzący dramatycznych skoków wartości na granicach. Poprzez modyfikacje współczynnika blendRange można ustalić szerokość obszaru blendowania.

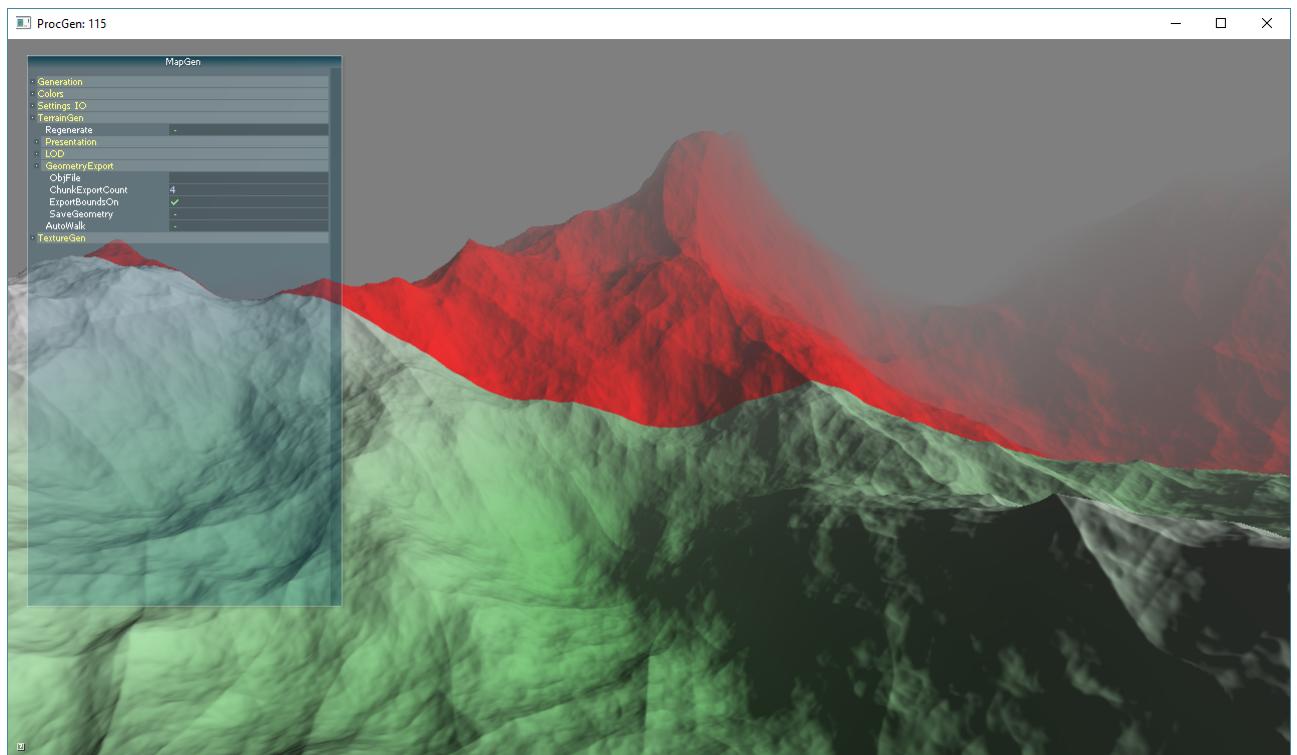
Pola podzakładki Noise Parameters:

- 1) NoiseType - Typ Szumu(Value, Perlin, Worley).
- 2) Frequency - Częstotliwość Bazowa.
- 3) Octaves - Liczba oktaw.

- 4) Lacunarity - Przyrost częstotliwości.
- 5) Persistence - Współczynnik przyrostu wag.
- 6) Scale - Mnożnik wartości końcowej szumu.
- 7) extraParam/extraParam2 - Pola na dodatkowe parametry zależne od rodzaju szumu.

## 4.2. Eksport geometrii siatki terenu oraz tekstur

Aplikacja pozwala na export geometrii do pliku obj.



Rys. 4.19. Eksport geometrii terenu z zaznaczoną opcją ExportBoundsOn (Teren o kolorze czerwonym nie jest eksportowany).

W zakładce TerrainGen->GeometryExport znajdują się 4 pola (Rys. 4.19).

- 1) ObjFile - Pole na nazwę pliku, do którego ma być zapisana siatka terenu.
- 2) ChunkExportCount - Promień Chunk'ów w którego zakresie zapisywana jest geometria.
- 3) ExportBoundsOn - Flaga binarna ułatwiająca zobaczenie, terenu który jest eksportowany.

- 4) SaveGeometry - Przycisk zapisujący.

W prawym górnym domyślnie znajduje się obecnie generowana tekstura. Program umożliwia zapis tej tekstury do pliku bmp. Opcje specyficzne dla generacji tekstury znajdują się w zakładce TextureGen w panelu bocznym.

W zakładce TerrainGen->GeometryExport znajduje się 6 pól.

- 1) FileName - Nazwa pliku do zapisu.
- 2) SaveFile - Przycisk do zapisu tekstury.
- 3) RegenerateTex - Przycisk wymuszający regenerację tekstury.
- 4) HideTexture - Przycisk ukrywający tekstury z okna programu.
- 5) ChunkRadius - Promień w którego zakresie znajdują się chunki mieszczące się na teksturze.
- 6) CoordinateMultiplier - Mnożnik zmiennych pozycji  $x$  i  $y$  (używanych w wyrażeniu).

Zakładka Colors jak nazwa wskazuje przechowuje definicje kolorów. By dodać kolor na koniec listy należy kliknąć przycisk AddColor.

Każdy kolor ma przypisane 4 pola.

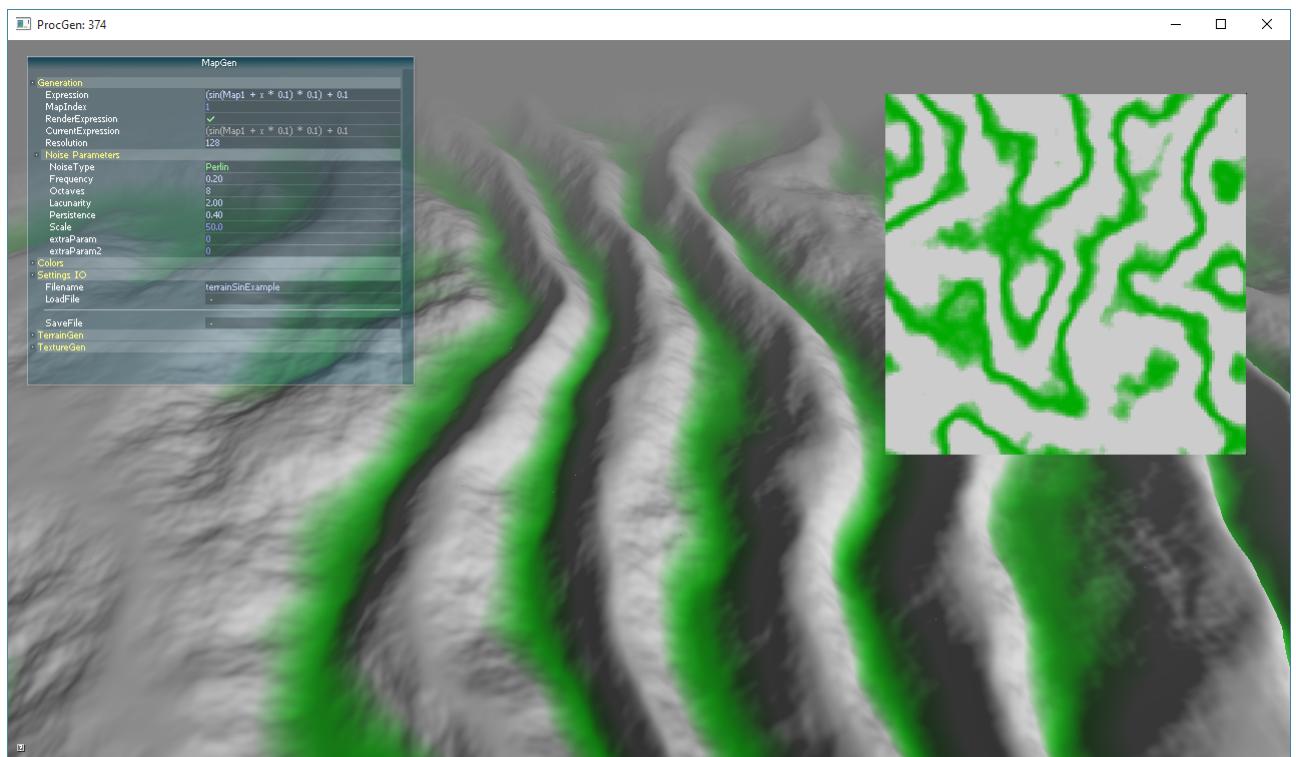
- 1) Color - Właściwy kolor.
- 2) StartValue - Wartość/Wysokość od której będzie zaczynał się ten kolor.
- 3) InsertColor - Przycisk, który dodaje domyślny kolor powyżej wybranego.
- 4) DeleteColor - Przycisk, który usuwa kolor z listy.

Kolor dla danego punktu jest obliczany poprzez interpolację liniową sąsiednich kolorów.

## 5. Metody tworzenia treści z wykorzystaniem szumów fraktalnych

### 5.1. Wykorzystanie podstawy sinusoidalnej

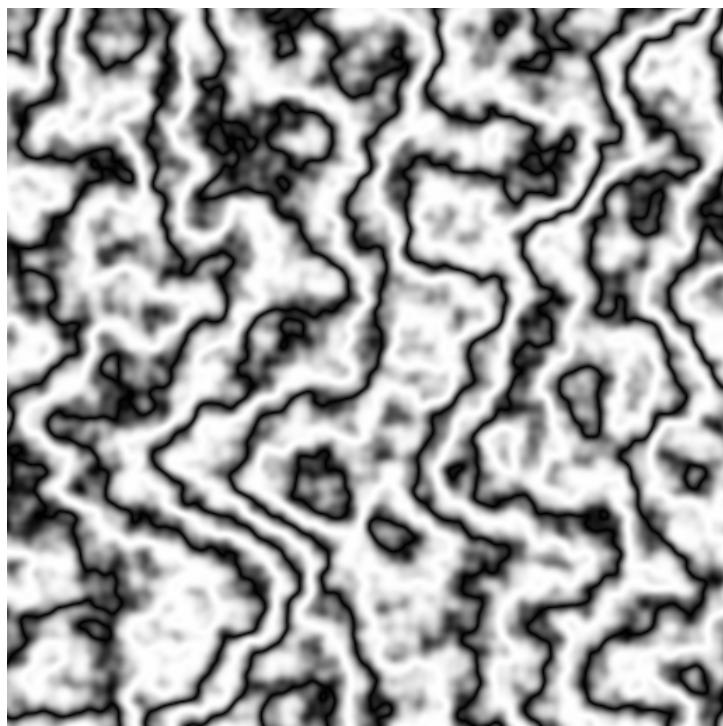
Wykorzystanie funkcji  $\sin$  z szumem fraktalnym. By utworzyć pasy wzgórz w danym kierunku można zastosować wyrażenie  $\sin(x) + 1.0$ . Po dodaniu do pozycji wartości szumu fraktalnego można otrzymać ciekawe regularne wzory - zakrzywione ciągłe 'wzniesienia'.



Rys. 5.20. Ilustracja zastosowania funkcji sin z szumem fraktalnym.

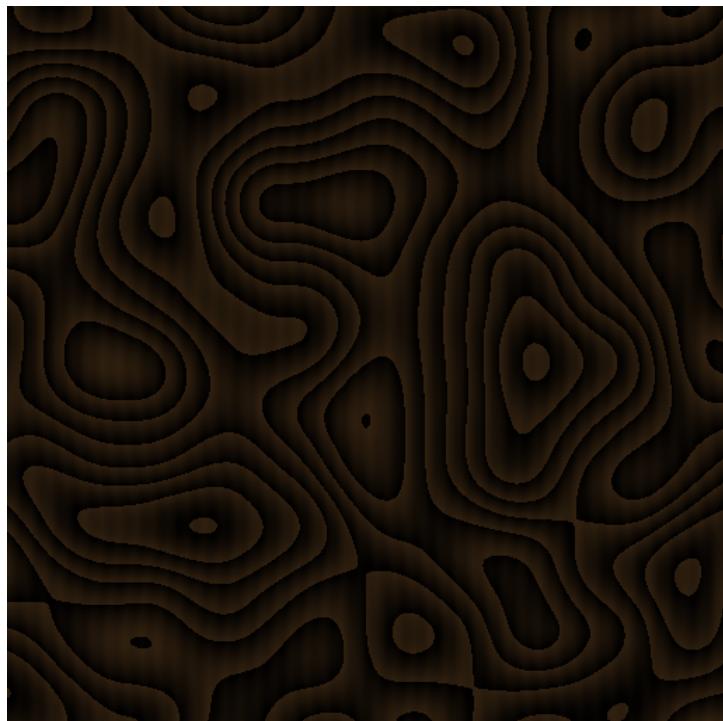
$$[(\sin(Map1 + x * 0.1) * 0.1) + 0.1]$$

Współczynnik skali wartości szumu fraktalnego Map1 określa stopień zwiększałczenia regularnych wartości sinusoidalnych. Taka podstawa może być użyta tworzenia tekstury marmuru, pionowych wzorów przy teksturowie deski czy wzorów piasku pustynnego.



Rys. 5.21. Wygenerowana tekstura marmuru.

$$[(\sin(Map1 + (x * 2.5)) + 1.0) * 0.5]$$

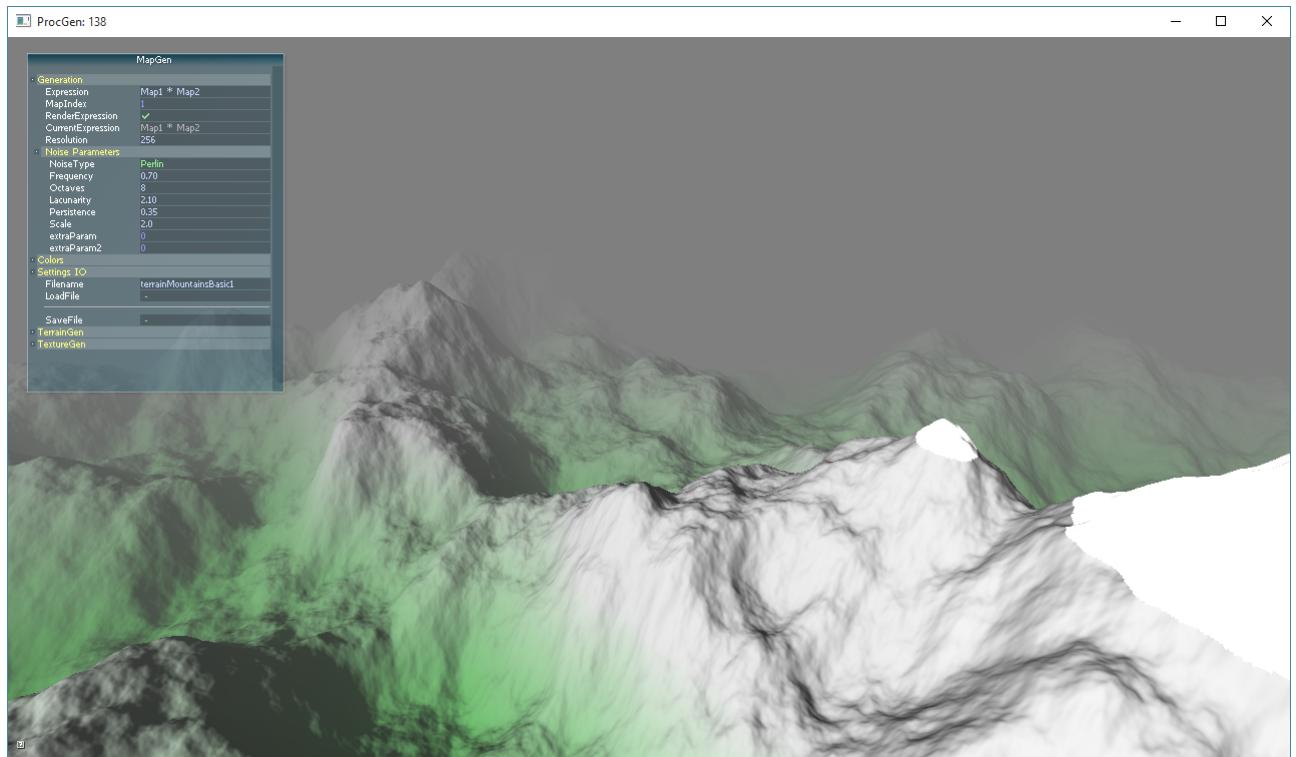


Rys. 5.22. Wygenerowana tekstura drewna.

$$[((\text{mod}(Map1, 0.1) * 10) * 0.9) + ((\sin((x * 30) + Map1 * 10) * 0.05) + 0.05)]$$

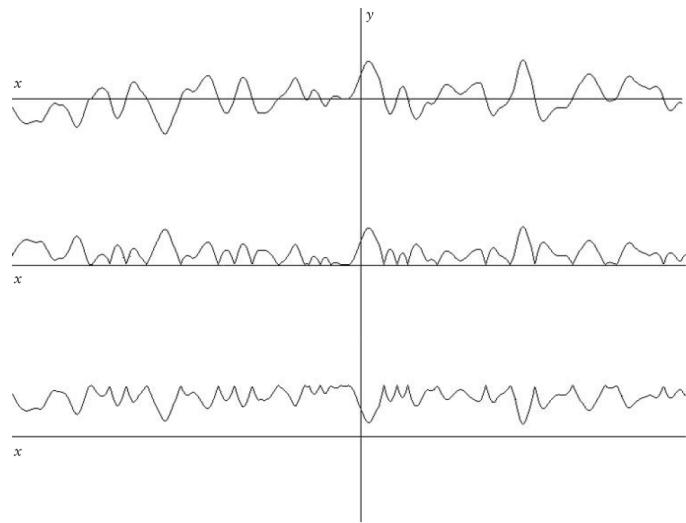
## 5.2. Ridged Perlin Noise

Standardowy szum perlina nie odwzorowuje stromych szczytów gór i charakterystycznych pasm dlatego nie jest odpowiedni do generacji typowo górzystych terenów (Rys. 5.23).

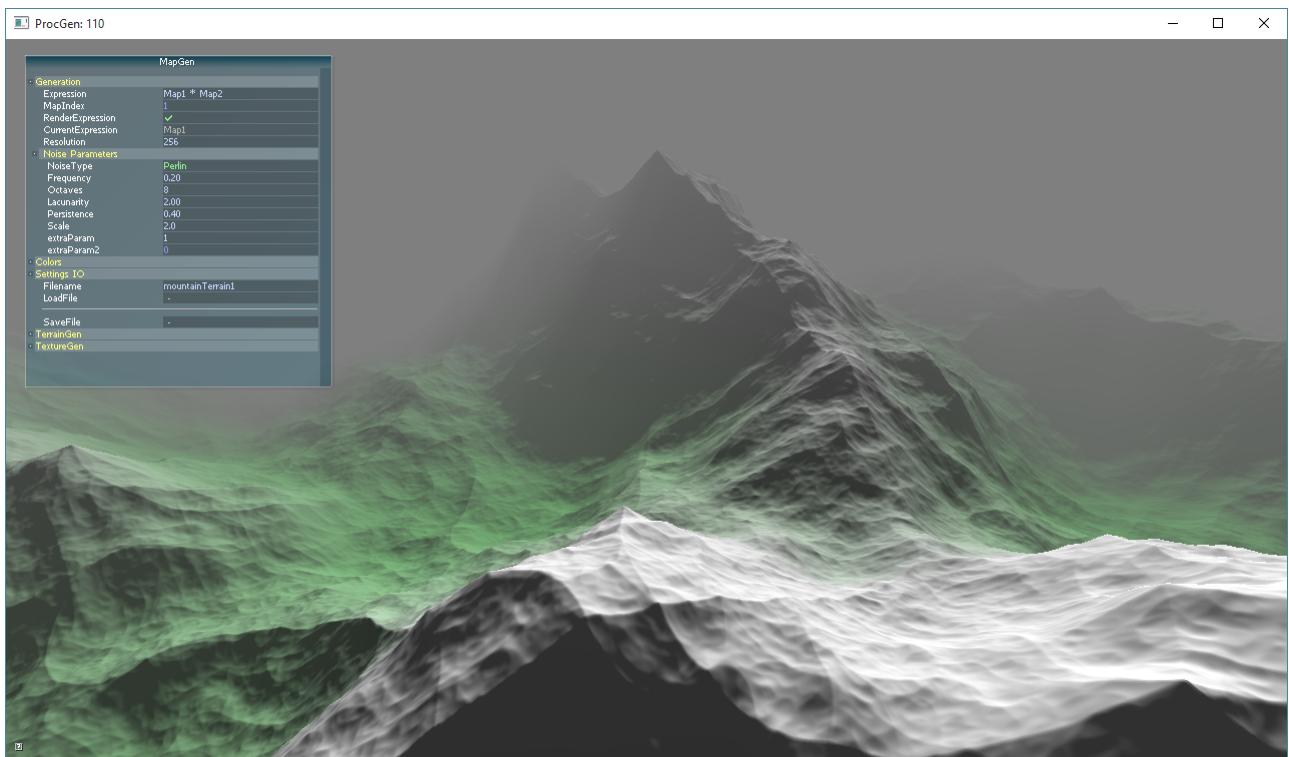


Rys. 5.23. Ilustracja przykładowej generacji gór z wykorzystaniem podstawowego szumu perlina.

Po prostej transformacji matematycznej szumu perlina można otrzymać rodzaj szumu fraktalnego, który dużo lepiej nadaje się do reprezentacji górzystego ukształtowania terenu. Jest nim ridged fractal noise.



Rys. 5.24. Ridged Noise. Dla każdej oktawy wartość szumu to  $(1.0 - abs(perlinVal))$ .



Rys. 5.25. Góry wygenerowane z wykorzystaniem Ridged Perlin Noise.

### 5.3. Wariacje szumu Worley'a

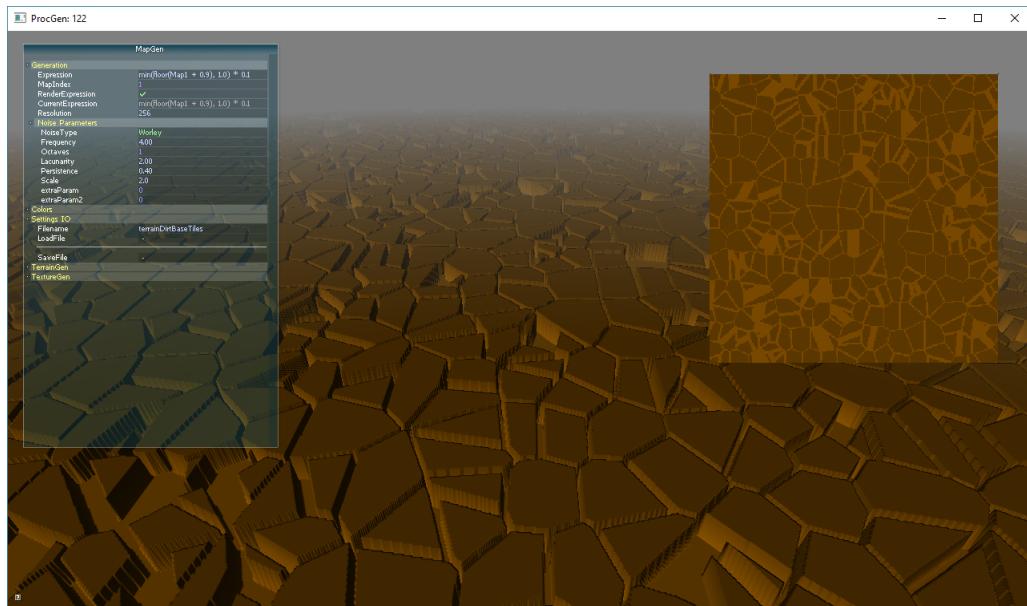
Ze względu na różne metryki odległość i klasyfikatory ( $f_1, f_2$  itp.), istnieje wiele kombinacji tworzących szum Worley'a.

Pole extraParam określa klasyfikatory odpowiednio:

- 1)  $0 - f_2 - f_1$ .
- 2)  $1 - f_1$ .
- 3)  $2 - f_2$ .
- 4)  $3 - f_2 + f_1$ .
- 5)  $4 - f_1 * f_2$ .

Pole extraParam2 określa metryki odległości odpowiednio:

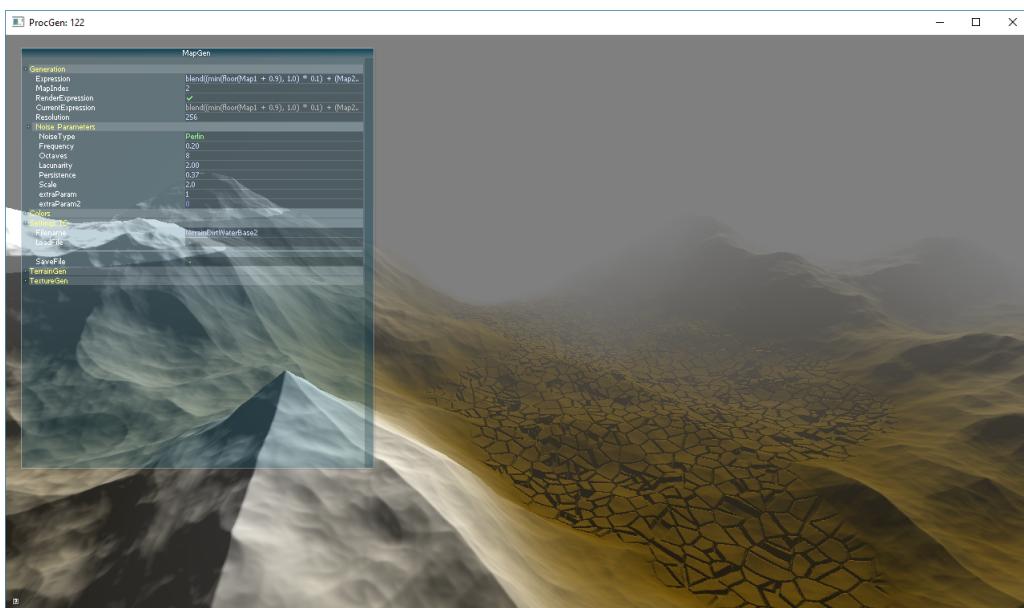
- 1) 0 - Euklidesowa kwadratowa ( $dx^2 + dy^2$ ).
- 2) 1 - Odległość manhattan ( $\text{abs}(dx) + \text{abs}(dy)$ )
- 3) 2 - Euklidesowa zwykła ( $\sqrt{dx^2 + dy^2}$ ).
- 4) 3 - Manhattan kwadratowa ( $(\text{abs}(dx) + \text{abs}(dy))^2$ ).
- 5) 4 - Odległość Czebyszewa Max ( $\max(dx, dy)$ ).
- 6) 5 - Odległość Czebyszewa Min ( $\min(dx, dy)$ ).



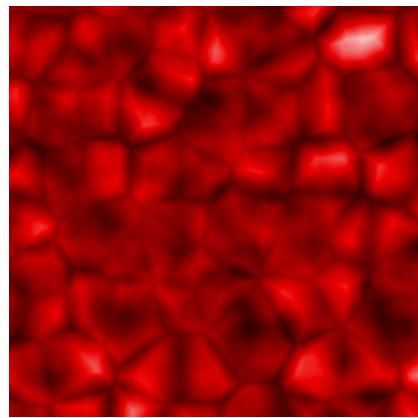
Rys. 5.26. Podstawowe kafelki, otrzymane poprzez użycie funkcji floor na szumie worley'a ( $f_2 - f_1$ , metryka Euklidesowa kwadratowa).



Rys. 5.27. Podstawowe kafelki, otrzymane poprzez użycie funkcji floor na szumie worley'a ( $f2 - f1$ , metryka Manhattan).



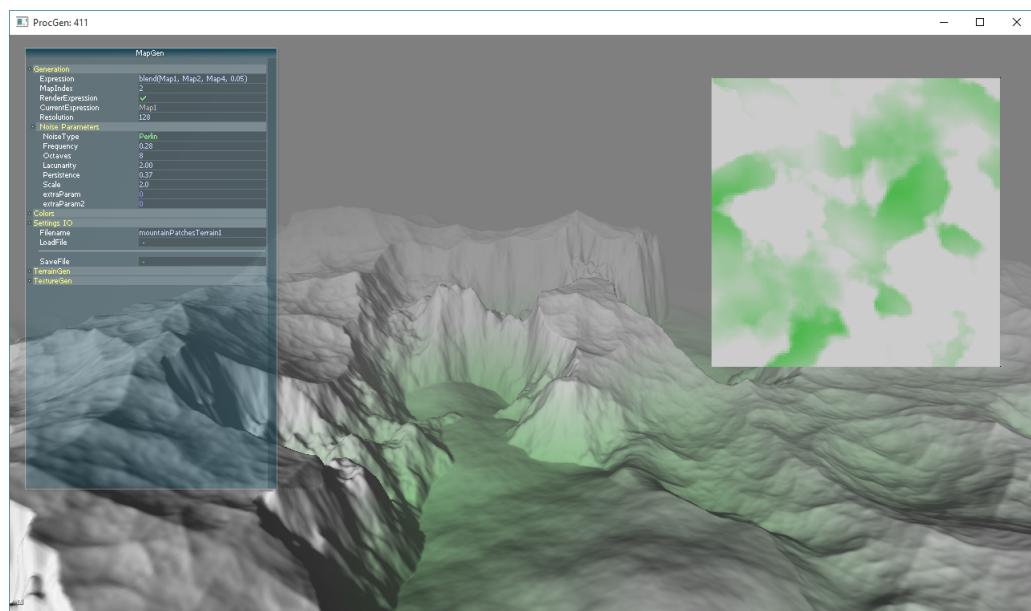
Rys. 5.28. Połączenie kafelków z ridged perlin noise - reprezentacja wysuszonego zbiornika wodnego.



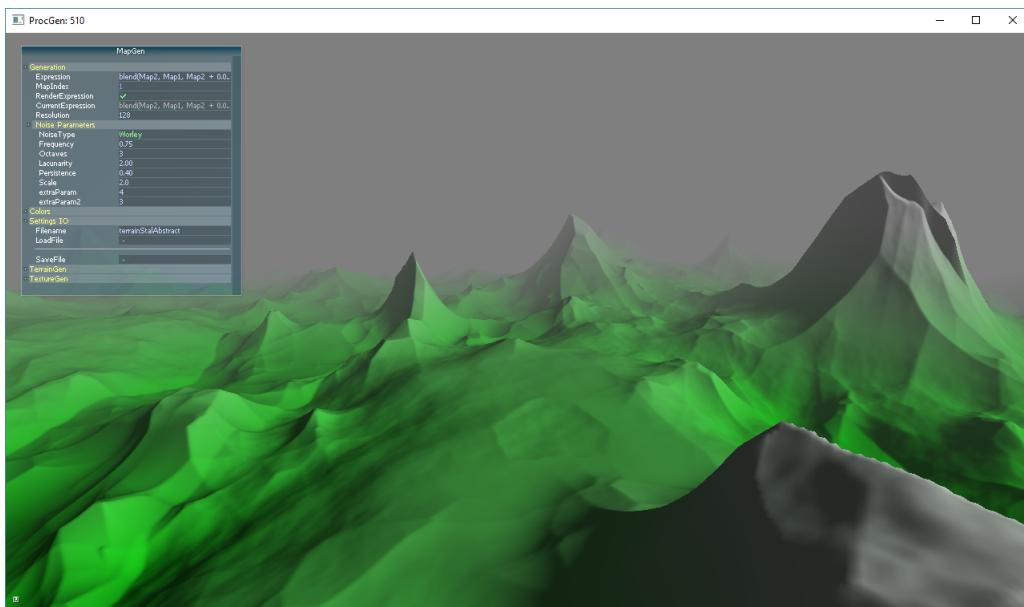
Rys. 5.29. Pokolorowane kafelki - worley( $f2 - f1$ ).

#### 5.4. Zastosowanie funkcji blend

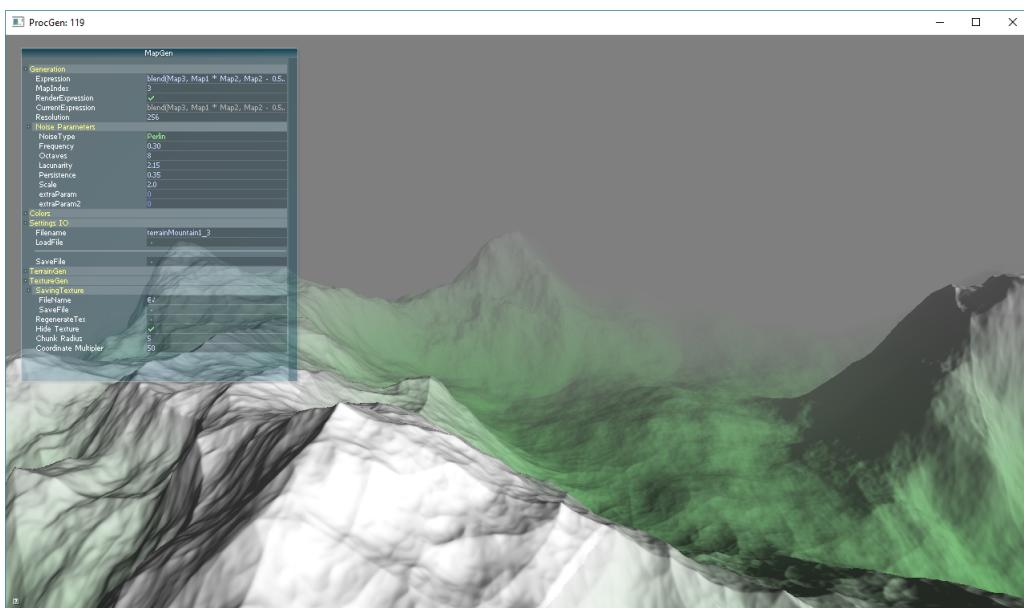
Połączenie dwóch szumów fraktalnych w funkcji trzeciego za pomocą funkcji *blend* pozwala na otrzymanie interesujących wzorów. Należy odnaleźć najczęściej po przez eksperymentacje 2 szumy fraktalne które w sąsiedztwie będą miały interesujący efekt.



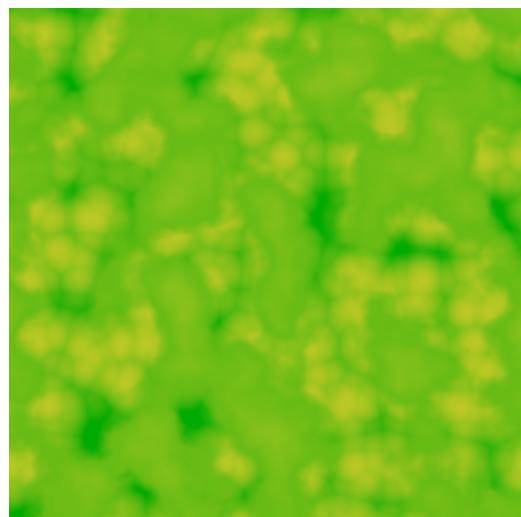
Rys. 5.30. Przykład wykorzystania funkcji *blend*



Rys. 5.31. Funkcja *blend* z wykorzystaniem ridged perlin noise oraz wieloktawowego szumu worley'a.



Rys. 5.32. Funkcja *blend* z wykorzystaniem ridged perlin noise oraz niskoczęstotliwościowego standardowego szumu perlin'a.



Rys. 5.33. Tekstura funkcji *blend* z wykorzystaniem szumu worley'a oraz perlin'a.

## 6. Podsumowanie i wnioski końcowe

Celem niniejszej pracy było przedstawienie sposobów generacji treści z wykorzystaniem szumów fraktalnych i utworzenie aplikacji ułatwiającej ten proces. Biorąc pod uwagę przykładowe rezultaty generacji zamieszczone w tej pracy, można uznać, że szумy fraktalne są przydatnymi konstrukcjami do generacji treści. W szczególnie efektywny sposób udało się odzwierciedlić charakter ostrych górzystych wzorów, z wykorzystaniem Ridged Perlin Noise i zasad multifraktalności.

Dzięki szumom można w prosty sposób eksperymentować z jakością treści, tworząc abstrakcyjne lub realistyczne wzory.

Wielowatkowość napisanego programu pozwala na szybką generację oraz działanie programu bez oczekiwania na zakończenie generacji, minimalizując tym samym czas pomiędzy iteracjami eksperymentów (zmiana wyrażenia, czy parametrów szumów). Podczas pracy nad programem udało się odkryć szereg nieprzewidzianie interesujących kombinacji szumów, poprzez czystą eksperymentację.

Jak się spodziewano szum Worley'a jest dużo bardziej wymagający obliczeniowo od pozostałych. By istotnie zwiększyć wydajność generacji korzystne będzie przekształcenie standardowego kodu C++ na taki wykorzystujący operacje wektorowe (SIMD - SSE) wspierane niemal przez każdy współczesny procesor.

Bardzo istotną zaletą proceduralnych metod jest możliwość regulacji rozdzielczości tworzonych treści, co ma zastosowanie w systemach LOD (Poziomu detali).

Z nakładem pracy nieznacznie przekrajającym wkład w program realizowany w tej pracy, można rozszerzyć generacje terenu do tworzenia różnych stref klimatycznych, czy nawet całych planet i galaktyk.

Jednakże prawdziwy potencjał szumu fraktalnego można zrealizować w kombinacji z innymi metodami proceduralnymi takimi jak l-systemy, implicit geometry czy synteza tekstur.

Autor za wkład własny uważa:

- Implementacje szumów (value, perlin'a, worley'a).
- Utworzenie aplikacji umożliwiającej użycie wyrażenia do generacji tekstur i terenu.
- Objasnienie kluczowych fragmentów kodu aplikacji.

- Opis wszystkich elementów interfejsu oraz dostępnych funkcjonalności.
- Opracowanie przykładowych zastosowań szumów do generacji treści przy pomocy utworzonej aplikacji.
- Utworzenie i implementacja funkcji *blend* dołączenia 2 szumów fraktalnych.
- Implementacja funkcjonalności odpowiedzialnej za prostą odczytywanie i zapisywanie predefiniowanych parametrów szumu oraz wyrażeń.

## Załączniki

## Literatura

- [1] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steven Worley :  
Texturing and Modeling a Procedural Approach.
- [2] <http://notch.tumblr.com/post/3746989361/terrain-generation-part-1>
- [3] <http://mrl.nyu.edu/~perlin/noise/>
- [4] [http://w3.marietta.edu/~biol/biomes/biome\\_main.htm](http://w3.marietta.edu/~biol/biomes/biome_main.htm)
- [5] <http://catlikecoding.com/unity/tutorials/noise/>
- [6] <http://devmag.org.za/2009/04/25/perlin-noise/>
- [7] [https://en.wikipedia.org/wiki/Value\\_noise](https://en.wikipedia.org/wiki/Value_noise)
- [8] <https://pl.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise>

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza  
Wydział Elektrotechniki i Informatyki  
Katedra Elektrotechniki i Podstaw Informatyki

Rzeszów, 2016

**STRESZCZENIE PRACY DYPLOMOWEJ INŻYNIERSKA  
WIZUALIZACJA I ZASTOSOWANIE SZUMU W GRAFICE  
KOMPUTEROWEJ**

Autor: Jakub Biel, nr albumu: EF-132678

Opiekun: dr inż. Grzegorz Drałus

Słowa kluczowe: szum, generacja, fraktalny, perlin, opengl

Celem pracy było przedstawienie sposobów generacji terenu, tekstur i transformacji z wykorzystaniem szumów fraktalnych. Praca obejmuje opracowanie teoretycznych zagadnień, szczegółowo implementacji algorytmów. Praktyczne zastosowanie poznanych metod do wygenerowania elementów o odpowiednich jakościach oraz opracowanie aplikacji ułatwiającej generację tych elementów w języku C++.

RZESZOW UNIVERSITY OF TECHNOLOGY  
Faculty of Electrical and Computer Engineering  
Department of Electrical and Computer Engineering  
Fundamentals

Rzeszow, 2016

**BSC THESIS ABSTRACT  
VISUALIZATION AND APPLICATION OF NOISE IN COMPUTER  
GRAPHICS**

Author: Jakub Biel, nr albumu: EF-132678

Supervisor: Grzegorz Drałus, PhD

Key words: noise, generation, fractal, perlin, opengl

The purpose of this thesis was to present the means of generating terrains, textures and transformations using fractal noise. The thesis spans the elaboration of theoretical issues, implementations of algorithms. Practical uses of known methods to generate elements of certain qualities as well as creating the application that facilitates generation of those elements in C++.