Analisis Perbandingan Perlin Noise Dan Simplex Noise Untuk Penciptaan Permukaan Daratan Pada Pembuatan Game

Wandi Wijaya[*1], Abdul Rahman[*2]
[1]STMIK GI MDP, [2]AMIK MDP; Jl. Rajawali No.14, +62(711)376400/376360
[1]Program Studi Teknik Informatika, [2]Program Studi Teknik Komputer
e-mail: [1]wandi.wijaya@mhs.mdp.ac.id, [2]arahman@mdp.ac.id

Abstrak

Procedural content generation merupakan salah satu cara untuk menciptakan objek atau perilaku yang bersifat acak serta otomatis, agar memiliki tampak yang berbeda dari objek lain. Content pada sebuah game seperti pembentukan permukaan daratan atau terrain dapat dibentuk secara unik menggunakan metode noise. Dalam sistem ini, noise yang akan digunakan adalah Perlin noise dan simplex noise. Kedua noise tersebut dikembangkan agar dapat membentuk suatu simulasi permukaan daratan yang menyerupai daratan, pegunungan, atau dasar laut. Hasil dari penelitian ini untuk mencari tahu tingkat efisiensi dan besarnya penggunaan beban komputasi di antara Perlin dan simplex noise. Metodologi yang digunakan terdiri dari proses studi literatur, analisa data, desain sistem, implementasi, dan pengujian sistem. Masin-masing noise memiliki bidang dimensi satu hingga dimensi tiga yang diuji berdasarkan input frekuensi noise, octave, persistence, lacunarity. Pengujian dilakukan dengan membandingkan penggunaan CPU, GPU, dan RAM komputer pada saat proses render masing-masing noise tersebut. Simplex noise yang berasal dari pengembangan Perlin noise tingkat lanjut memiliki tingkat efisiensi penggunaan CPU, GPU, dan RAM yang lebih baik daripada Perlin noise.

Kata kunci: *Permukaan daratan, Terrain, Noise, Perlin, Simplex*

1. Pendahuluan

Desain dan rancangan sebuah peta merupakan bagian penting pada berbagai macam game seperti first-person shooter (FPS), real-time strategy (RTS), dan role playing game (RPG). Terrain generation atau proses penciptaan permukaan daratan adalah salah satu contoh implementasi procedural content generation (PCG) dalam industri game saat ini. Metode procedural content generation telah digunakan oleh perindustrian game selama tiga decade untuk membangkitkan desain dan konten pada sebuah game, baik secara semi otomatis (manusia dengan komputer) atau diproses menyeluruh secara otomatis oleh komputer [1]. Konten-konten yang dimaksud dapat berupa level, misi pada permainan, tekstur visual, karakter, vegetasi, beberapa set aturan, dan dinamika bangunan serta permukaan daratan.

Ada beberapa alasan mengapa metode PCG perlu diterapkan pada pengembangan sebuah game [2]. Pertama, permainan menjadi lebih menantang dan menarik minat pemain karena desain level atau konten yang selalu berbeda. Kedua, ada beberapa permainan yang membutuhkan real-time generation dan infinite level seperti Subway Surfer di mana letak jalan dan rintangan selalu berbeda untuk setiap sesi permainan. Ketiga, kerumitan dan penyesuaian rancangan peta dapat dibuat berdasarkan perilaku pemain saat bermain game tersebut. Keempat, metode PCG dapat digunakan sebagai suatu alat yang dapat melengkapi kreativitas manusia saat merancang desain suatu peta.

Dalam menerapkan metode PCG ini, penulis akan menerapkan dua algoritma dalam PCG yaitu algoritma Perlin noise dan simplex noise. Noise-based procedural content telah digunakan untuk mensimulasikan awan, badai, api, riak dan ombak air, serta daratan pegunungan [3]. Survei yang dilakukan Smelik [4] menyatakan bahwa proses penciptaan daratan banyak yang menggunakan Perlin noise yang ditemukan Ken Perlin di tahun 1985. Simplex noise sendiri merupakan pengembangan lanjut dari Perlin noise di tahun 2001 yang memiliki fundamental yang sama namun memiliki struktur komputasi yang berbeda [5].

Berdasarkan pemaparan di atas dalam penelitian ini akan menerapkan Perlin noise dan simplex noise untuk menciptakan permukaan daratan yang berbasis procedural content generation. Kedua noise ini

akan dikomputasi kedalam citra yang disebut height map sebagai tempat menyimpan informasi ketinggian yang nanti dapat digunakan dalam pengolahan geometri 3D. Melalui height map inilah simulasi permukaan daratan dapat tercipta. Selain itu, penulis juga akan menguji performa dari kedua noise tersebut dari segi rendering dan pemakaian resource komputer mana yang lebih baik dan efisien.

2. Metode Penelitian

Metodologi yang digunakan dalam pengembangan sistem ini merupakan metodologi yang terdiri dari beberapa tahapan seperti:

1. Studi Literatur

   Di tahapan ini merupakan melakukan analisa terhadap *hardware* dan *software* yang dibutuhkan dalam mengembangkan *Perlin* dan *simplex noise* serta pengumpulan informasi dan teori dengan membaca beberapa buku dan jurnal untuk mendukung proses pengembangan tersebut.

2. Analisa Data

   Setelah mengumpulkan referensi, proses selanjutnya adalah melakukan analisa terhadap proses interpolasi pada *Perlin noise* dan proses *skewing* pada *simplex noise*. Analisa berikutnya adalah bagaimana menerapkan *noise* tersebut pada dimensi vektor yang berbeda-beda mulai dari dimensi satu hingga dimensi tiga.

3. Desain Sistem

   Pada tahapan ini beberapa rancangan yang akan digunakan pada aplikasi, berupa perancangan *user editor* untuk mengontrol pembentukan *noise* dan pembuatan alur kerja *noise* yang dijelaskan di dalam *flowchart*.

4. Implementasi Sistem

   Proses implementasi dimulai dengan membuat satu objek *mesh* kosong dengan ukuran 10x10. Melalui bidang tersebut, langkah berikutnya adalah menentukan jarak vektor pada bidang tersebut agar *noise* dapat memenuhi bidang tersebut. Bentuk *noise* berbeda sesuai dengan posisi koordinat vektor, dimensi, parameter input serta jenis *noise* apakah *Perlin* atau *simplex noise*. Tinggi permukaan daratan yang dibentuk disesuaikan oleh skala ketinggian.

5. Pengujian Sistem

Tahapan terakhir adalah pengujian sistem *Perlin noise* dan *simplex noise* dalam mensimulasikan permukaan daratan. Pengujian *noise* ini ditentukan dengan berbagai variasi frekuensi, *octave*, *lacunarity*, *persistence* gelombang *noise* yang dipilih serta dimensi *noise* pada koordinat tertentu. Proses pencatatan beban komputasi komputer akan dilakukan saat pertama kali aplikasi melakukan *render* terhadap bentuk *noise*. Pengujian dilakukan berulang kali mulai dari besaran input kecil hingga besar.

3. Hasil dan Pembahasan

3.1 Perlin Noise

Perlin noise dikembangkan oleh Ken Perlin di tahun 1985 sebagai teknik penghalusan (*smoothing*) pada *noise* yang pada waktu itu terlalu kasar untuk digunakan pada proses penciptaan *terrain*. *Perlin noise* juga dikenal sebagai *gradient noise* di mana titik-titik gradien diatur secara *pseudo-random* pada sebuah ruang dan terjadi proses interpolasi dan penghalusan diantara titik-titik tersebut [6].

Rumus perhitungan interpolasi *Perlin noise* adalah sebagai berikut:

$$P = (x, y), i = floor\,(x), j = floor(y)\ (1)$$

$$g_{00} = gradient\ at\ (i, j), \qquad g_{10} = gradient\ at\ (i+1, j)\ (2)$$

$$g_{01} = gradient\ at\ (i, j+1), \qquad g_{11} = gradient\ at\ (i+1, j+1)\ (3)$$

$$u = x - i, v = y - j\ (4)$$

$$n_{00} = g_{00} \cdot \begin{bmatrix} u \\ v \end{bmatrix}, n_{10} = g_{10} \cdot \begin{bmatrix} u-1 \\ v \end{bmatrix}, n_{01} = g_{01} \cdot \begin{bmatrix} u \\ v-1 \end{bmatrix}, n_{11} = g_{11} \cdot \begin{bmatrix} u-1 \\ v-1 \end{bmatrix}\ (5)$$

$$n_{x0} = n_{00}\big(1 - f(u)\big) + n_{10} f(u), n_{x1} = n_{01}\big(1 - f(u)\big) + n_{11} f(u)\ (6)$$

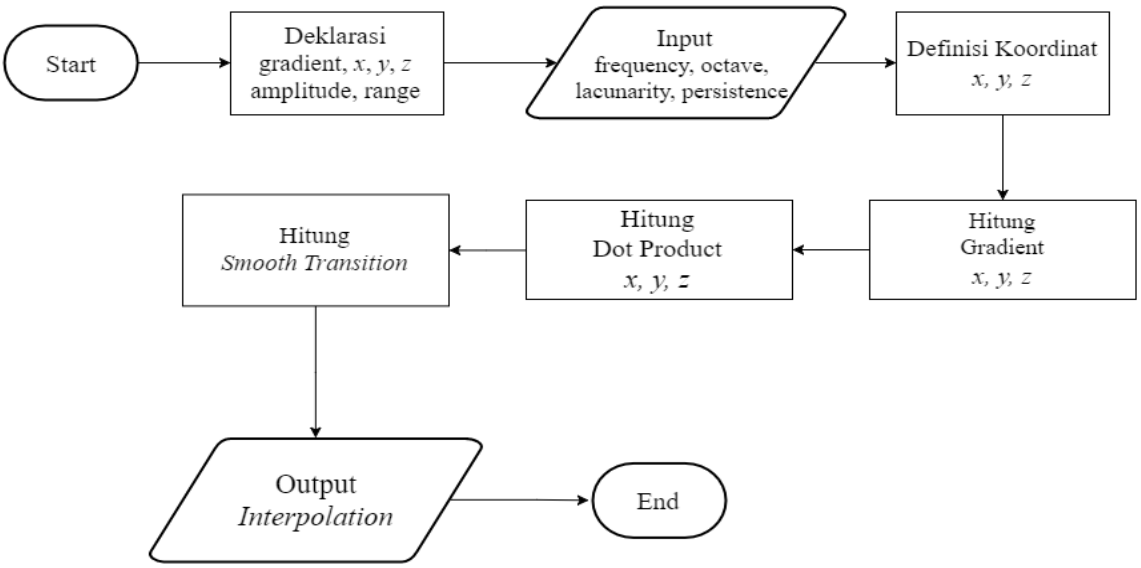
$$n_{xy} = n_{x0}\big(1 - f(v)\big) + n_{x1} f(v)\ (7)$$

Pada *Flowchart Pemrosesan Perlin noise* Gambar 1, proses dimulai dengan deklarasi nilai gradien, vektor pada sumbu *x, y,* dan *z*. Setelah itu, dengan menerima *input frequency*, *lacunarity*, *persistence*, dan *octave*. Maka proses selanjutnya adalah mendefiniskan koordinat *x, y*, dan *z*.

Untuk *Perlin noise* dimensi satu, koordinat vektor yang digunakan diambil dari sumbu (*x)*, dimensi dua menggunakan sumbu (*x, y*), dan dimensi tiga menggunakan sumbu (*x, y, z*). Vektor *x, y, z* dari dimensi yang dipilih kemudian dicari nilai vektor gradiennya. Nilai ini kemudian dihitung secara *dot product* dengan vektor jarak (*x, y, z*) agar dapat menentukan posisi *noise* pada tiap sumbu (*x, y, z*). Jumlah vektor gradien dan vektor jarak yang dibutuhkan bergantung pada fungsi 2^D, di mana *D* adalah jumlah dimensi. Setelah mendapatkan nilai *dot* maka langkah berikutnya adalah memperhalus *noise* pada tiap koordinat dengan fungsi:

$$f(t) = 6t^5 - 15t^4 + 10t^3 \ (8)$$

Langkah terakhir yaitu melakukan proses interpolasi antara *dot product* dan nilai $f(t)$.



Gambar 1 *Flowchat* Pemrosesan *Perlin Noise*

3.2 Simplex Noise

Simplex noise merupakan pengembangan dari *Perlin noise* di mana *noise* ini memiliki skala dimensi yang lebih tinggi (4-dimensi ke atas) dengan proses komputasi yang lebih ringan. Komputasi yang lebih rendah ini dikarenakan titik-titik pada *simplex noise* memiliki jarak sudut dimensi yang lebih dekat untuk mereferensikan setiap *pixel*. Bentuk *simplex* pada 2D berbentuk segitiga, 3D berbentuk piramid, dan untuk dimensi-n memiliki sudut *n + 1* [7].

Proses implementasi *simplex noise* adalah:

1. Menghitung koordinat kemiringan

$$F = \frac{\sqrt{n+1} - 1}{n} \ (1)$$
$$x' = x + (x + y + \cdots) * F \ (2)$$
$$y' = y + (x + y + \cdots) * F \ (3)$$

Melalui perhitungan dilakukan untuk mencari nilai dua sudut x dan y dengan membulatkan hasil koordinat ke atas ataupun ke bawah. Jika nilai relatif x' lebih besar daripada nilai relatif y', maka titik koordinat *pixel* berada di bagian kanan bawah segitiga, dan nilai sudut didapatkan dengan membulatkan nilai *x'* ke atas dan nilai *y'* ke bawah. Sebaliknya bila nilai relatif *y'* lebih besar daripada nilai relatif *x'*, maka titik koordinat *pixel* berada di bagian atas kiri segitiga dan nilai sudut didapatkan dengan membulatkan nilai *x'* ke bawah dan nilai *y'* ke atas.

2. Mengembalikan nilai koordinat kemiringan menjadi normal

$$G = \frac{n + 1 - \sqrt{n+1}}{n * (n+1)} \quad (4)$$
$$x = x' + (x' + y' + \cdots) * G \quad (5)$$
$$y = x' + (x' + y' + \cdots) * G \quad (6)$$

Setelah menemukan titik koordinat dalam sudut ruang miring, maka perhitungan kemiringan harus dikembalikan lagi ke *simplex* biasa untuk menentukan jarak normal tersebut.

3. Perhitungan *dot product* untuk setiap sudut

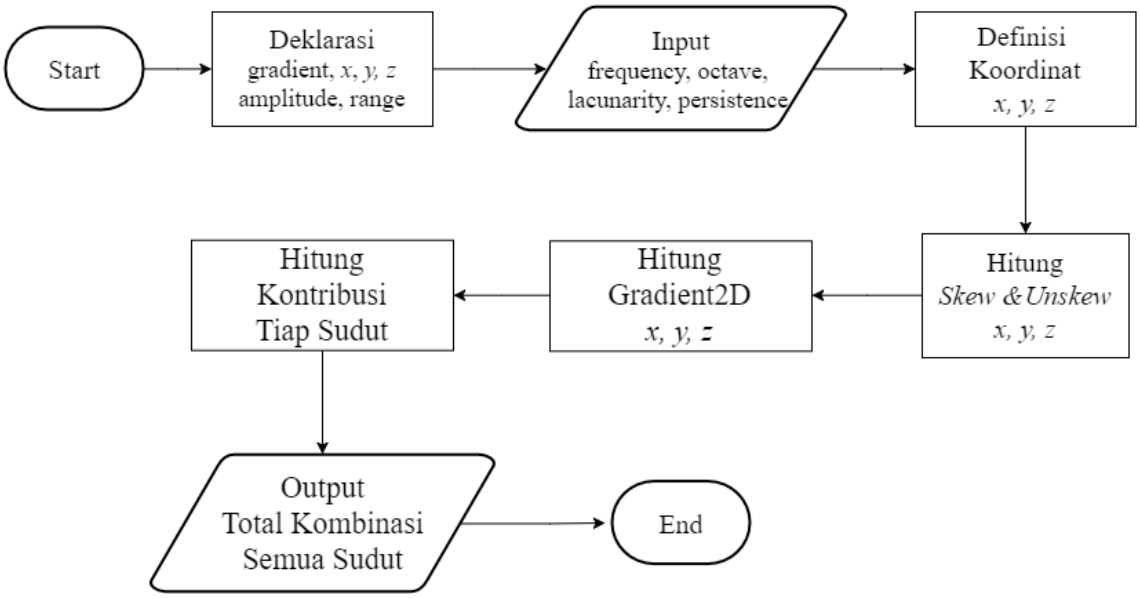
$$radius = \min(0.0,\ 0.5 - pow(x_{distance}, 2) - pow\left(y_{distance,}2\right) - \cdots \quad (7)$$
$$Hasil\ perhitungan\ radial = pow(radius, 4) * dot_{productresult} \quad (8)$$

Pada gambar *Flowchart Pemrosesan Simplex Noise* di bawah, proses dimulai dengan deklarasi nilai gradien, vektor pada sumbu *x, y,* dan *z*. Setelah itu, dengan menerima *input frequency*, *lacunarity*, *persistence*, dan *octave*. Maka proses selanjutnya adalah mendefiniskan koordinat *x*, *y*, dan *z*.

Untuk *simplex noise* dimensi satu, koordinat vektor yang digunakan diambil dari sumbu (*x)*, dimensi dua menggunakan sumbu (*x, y*), dan dimensi tiga menggunakan sumbu (*x, y, z*). Setelah menentukan vektor (*x, y, z*), langkah berikutnya adalah menentukan *skew* atau kemiringan posisi (*x*), (*x, y*), atau (*x, y, z*) dengan mengkalikan nilai vektor tersebut dengan derajat kemiringan. Setelah menghitung nilai kemiringan vektor yang telah berubah tersebut, nilai kemiringan akan dikembalikan kembali (*unskew*) untuk mendapatkan jarak vektor baru untuk setiap koordinat *x, y,* dan *z*.

Melalui vektor-vektor tersebut, langkah berikutnya adalah menentukan posisi *simplex* dan sudut mana pada tiap koordinat. Jumlah sudut ditentukan dari $n + 1$, di mana n merupakan dimensi *simplex*. Jumlah sudut-sudut tersebut digabung menjadi satu untuk mendapatkan nilai utuh *simplex noise* pada suatu bidang.

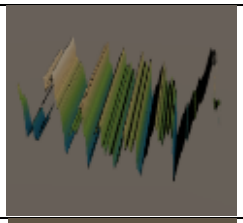
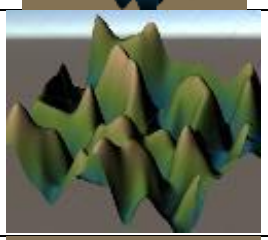
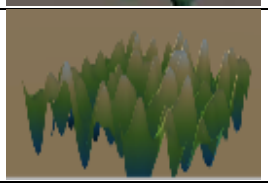
Vektor *x, y, z* dari dimensi yang dipilih kemudian dicari nilai vektor gradiennya. Nilai ini kemudian dihitung secara *dot product* dengan vektor jarak (*x, y, z*) agar dapat menentukan posisi *noise* pada tiap sumbu (*x, y, z*). Langkah terakhir yaitu melakukan proses interpolasi antara *dot product* dan nilai $f(t)$.



Gambar 2 *Flowchart* Pemrosesan *Simplex Noise*

3.3 *Analisis Hasil Pengujian*

Tabel 1 Hasil Pengujian *Perlin* dan *Simplex Noise*

Dimensi	Perilaku	*Noise*	CPU	GPU	RAM	Gambar
1	*Frequency* : 7 *Octave* : 6 *Lacunarity* : 3 *Persitence* : 0.4	***Perlin***	**40.70 ms**	**1386.07 ms**	**215.2 MB**	
		Simplex	**47.85 ms**	**1320.41 ms**	**214.1 MB**	
2	*Frequency* : 6 *Octave* : 3 *Lacunarity* : 1.5 *Persitence* : 0.24	***Perlin***	**39.66 ms**	**1347.86 ms**	**215.4 MB**	
		Simplex	**44.90 ms**	**1359.89 ms**	**216.6 MB**	
3	*Frequency* : 6 *Octave* : 3 *Lacunarity* : 1.5 *Persitence* : 0.24	***Perlin***	**73.32 ms**	**1524.47 ms**	**215.3 MB**	
		Simplex	**49.51 ms**	**1301.54 ms**	**216.3 MB**	

Tabel 1 merupakan hasil pengujian *perlin noise* dan *simplex noise* untuk perilaku *frequency*, *octave*, *lacunarity*, dan *persistence* yang sama untuk dimensi satu, dua, dan tiga. Satuan CPU dan GPU untuk lamanya proses *render* diukur dalam milisekon (ms) sedangkan satuan penggunaan memori RAM dalam *megabyte* (MB).

Dari data pada tabel 2 dan 3, secara rata-rata penggunaan CPU dan GPU untuk *simplex noise* lebih efisien daripada *Perlin noise*. Penggunaan RAM untuk masing-masing pengujian dimensi secara garis besar hampir sama meski penggunaan CPU dan GPU lebih fluktuatif pada *simplex noise*. Namun secara rata-rata, penggunaan RAM pada *simplex noise* lebih baik daripada *Perlin noise*.

Tabel 2 Tabel Penggunaan Daya Komputer untuk *Perlin Noise*

Jumlah Pengujian	*Perlin* Dimensi ke-	Jumlah Pemakaian			Standar Deviasi		
		CPU (ms)	GPU (ms)	RAM (MB)	**CPU**	**GPU**	**RAM**
10	1D	358.67	13448.35	2182.8	**11.50**	**110.62**	**6.22**
	2D	587.26	14932.76	2198.2	**18.86**	**105.39**	**7.21**
	3D	943.68	15007.37	2185.1	**31.67**	**113.72**	**6.32**

Tabel 3 Tabel Penggunaan Daya Komputer untuk *Simplex Noise*

Jumlah Pengujian	*Simplex* Dimensi ke-	Jumlah Pemakaian			Standar Deviasi		
		CPU (ms)	GPU (ms)	RAM (MB)	**CPU**	**GPU**	**RAM**
10	1D	393.4	12970.28	2171.3	**13.15**	**76.35**	**6.42**
	2D	668.03	14354.18	2165.5	**22.97**	**77.69**	**0.206**
	3D	739	14044.45	2176.2	**22.54**	**101.17**	**5.06**

Dari hasil analisis pengujian *Perlin noise* dan *simplex noise*, dengan penerapan *noise* menggunakan ruang satu sampai tiga dimensi maka dapat disimpulkan bahwa *simplex noise* memiliki proses simulasi permukaan daratan yang lebih cepat dan efisien daripada *Perlin noise*, dilihat dari penggunaan CPU, GPU, dan RAM.

4. Penutup

4.1. Kesimpulan

Dari hasil penelitian yang telah dilakukan maka kesimpulan yang diperoleh dari pengujian kedua noise ini adalah:

1. Penerapan Perlin noise dan simplex noise untuk menciptakan atau mensimulasikan permukaan daratan dapat dilakukan pada noise dimensi dua dan dimensi tiga.
2. Perlin noise merepresentasikan bentuk permukaan daratan lebih baik daripada simplex noise jika dibandingkan berdasarkan pada frequency, octave, lacunarity, dan persistence yang sama.
3. Dari analisis hasil pengujian Perlin noise dan simplex noise, dengan membandingkan beban komputasi pada saat proses render, terbukti bahwa simplex noise memiliki waktu pemrosesan dan tingkat efisiensi penggunaan CPU, GPU, dan RAM yang lebih baik daripada Perlin.

4.2. Saran

Saran yang dapat direkomendasikan oleh penulis dalam penelitian kali ini adalah:

1. Penggunaan metode-metode lain dalam simulasi penciptaan permukaan daratan untuk mengetahui perbandingan penggunaan CPU, GPU, dan RAM antara masing-masing metode tersebut.
2. Penggunaan material dan tekstur daratan sesungguhnya seperti rumput, pasir, permukaan gunung yang kasar, dan lain-lain.
3. Cakupan area noise untuk permukaan daratan lebih luas sehingga dapat digunakan pada game yang bersifat open-world atau ekplorasi.

Daftar Pustaka

[1] Smith, AJ & Joanna JB 2014, A Logical Approach to Building Dungeons: Answer Set Programming for Hierarchical Procedural Content Generation in Roguelike Games, University of Bath.
[2] Johnson, L, Georgios NY & Julian T 2010, Cellular Automata for Real-Time Generation of Infinite Cave Levels, IT University of Copenhagen.
[3] Spjut, JB dkk 2009, Hardware-Accelerated Gradient Noise for Graphics, Proceedings of the 19th ACM Great Lakes symposium on VLSI, Vol. 8, h. 457-462.
[4] Smelik, RM dkk 2009, A Survey of Procedural Methods for Terrain Modelling, Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation, Vol.1, h. 25-34.
[5] McEwan, I dkk 2012, Efficient Computational Noise in GLSL, Journal of Graphics Tools, Vol.16, No.2, h.85-94.
[6] Gustavon, S 2005, Simplex Noise Demystified, Lingkoping University, Lingkongping.