TUGAS AKHIR – IF184802

Optimasi Proses Pembangkitan Lanskap Secara *Real-time* Menggunakan Paralel *Procedural Content Generation* Pada Permainan DigiTale

FIRMAN MAULANA
05111640000059

Dosen Pembimbing I :
Dr.Eng.Darlis Herumurti S.Kom., M.Kom.

Dosen Pembimbing II :
Hadziq Fabroyir S.Kom., Ph.D.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2020

TUGAS AKHIR – IF184802

Optimasi Proses Pembangkitan Lanskap Secara *Real-time* Menggunakan Paralel *Procedural Content Generation* Pada Permainan DigiTale

FIRMAN MAULANA
05111640000059

Dosen Pembimbing I :
Dr.Eng.Darlis Herumurti S.Kom., M.Kom.

Dosen Pembimbing II :
Hadziq Fabroyir S.Kom., Ph.D.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2020

[*Halaman ini sengaja dikosongkan*]

FINAL PROJECT – IF184802

Optimizing the Landscape Generation Process in Real-time Using Parallel Procedural Content Generation in DigiTale Games

FIRMAN MAULANA
05111640000059

Supervisor I
Dr.Eng.Darlis Herumurti S.Kom., M.Kom.

Supervisor II
Hadziq Fabroyir S.Kom., Ph.D.

DEPARTMENT OF INFORMATICS ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya
2020

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

Optimasi Proses Pembangkitan Lanskap Secara *Real-time* Menggunakan Paralel *Procedural Content Generation* Pada Permainan *DigiTale*

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Grafika, Interaksi dan Game
Program Studi S-1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh:
FIRMAN MAULANA
NRP: 051116 40000 059

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Dr.Eng.Darlis Herumurti S.Kom., M.Kom.
NIP. 197712172003121001 ……
(Pembimbing 1)

Hadziq Fabroyir S.Kom., Ph.D.
NIP. 198602272019031006 ……
(Pembimbing 2)

SURABAYA
2020

[*Halaman ini sengaja dikosongkan*]

Optimasi Proses Pembangkitan Lanskap Secara *Real-time* Menggunakan Paralel *Procedural Content Generation* Pada Permainan *DigiTale*

Nama Mahasiswa	:	Firman Maulana
NRP	:	051116 40000 059
Departemen	:	Teknik Informatika FTEIC - ITS
Dosen Pembimbing 1	:	Dr.Eng.Darlis Herumurti S.Kom., M.Kom.
Dosen Pembimbing 2	:	Hadziq Fabroyir S.Kom., Ph.D.

Abstrak

Aplikasi permainan merupakan salah satu media yang dapat digunakan untuk sarana hiburan. Sarana hiburan yang satu ini berhubungan erat dengan konten yang ada. Semakin banyak konten pada aplikasi permainan ini maka semakin lama waktu yang dibutuhkan oleh pemain untuk menyelesaikan permainan ini. Salah satu metode untuk memperbanyak konten adalah dengan menerapkan Procedural Content Generation. Namun dengan menggunakan ini maka dibutuhkan kemampuan pemrosesan yang cukup tinggi. Aplikasi permainan juga merupakan salah satu jenis perangkat lunak yang harus diproses secara real-time. Pemrosesan secara real-time ini membutuhkan strategi tertentu yang tidak hanya berbasis pada kecepatan pemrosesan. Dengan menggunakan strategi pemrosesan yang tepat maka dapat memberikan kesan kepada pemain bahwa aplikasi ini berjalan dengan lancar. Penggunaan procedural content generation secara real-time akan memakan waktu yang cukup tinggi dan membutuhkan metode pemrosesan tersendiri.

Pada tugas akhir ini akan dibuat sebuah aplikasi permainan yang memanfaatkan procedural content generation untuk pembuatan lanskap secara real-time. Tugas akhir ini akan memanfaatkan kemampuan prosesor untuk memproses data secara

paralel. Tugas akhir ini juga mengatasi masalah dimana suatu tugas tidak dapat dijalankan secara parallel namun tetap responsif dimata pengguna. Hal ini diraih dengan cara membagi waktu pemrosesan yang boleh dilakukan di prosesor utama.

Pengujian dilakukan dengan cara melakukan profiling terhadap aplikasi permainan ini untuk mengetahui kemampuan aplikasi dalam melakukan pemrosesan tanpa mengurangi kenyaman pengguna. Hasil yang diperoleh ketika menggunakan metode yang diajukan memberikan waktu pemrosesan dari 18 milidetik hingga 29.87 milidetik dibandingkan ketika tanpa menggunakannya yaitu sebesar 210 milidetik hingga 243 milidetik.

Kata kunci: Pemrosesan, Permainan, *Real-time*, *Procedural Content Generation*

Optimizing the Landscape Generation Process in Real-time Using Parallel Procedural Content Generation in DigiTale Games

Student Name	: Firman Maulana
Registration Number	: 051116 40000 059
Department	: Informatics Engineering Department Faculty of Intelligent Electrical and Informatics Technology – ITS
First Supervisor	: Dr.Eng.Darlis Herumurti S.Kom., M.Kom.
Second Supervisor	: Hadziq Fabroyir S.Kom., Ph.D.

Abstract

The game application is one of the media that can be used for entertainment. This entertainment facility is closely related to the content provided. The more content in a game application, the longer the time needed by players to complete a game. One method for increasing content is to apply the Procedural Content Generation. However, by using this method, a high processing capability is needed. Game applications are also a type of software that must be processed in real-time. This real-time processing requires a certain strategy that is not only based on processing speed but also correct strategy. By using the right processing strategy, it can give the impression to the player that the application is running smoothly. The use of procedural content generation in real-time will be quite a time consuming and requires a separate processing method.

In this final project, a game application will be made that utilizes procedural content generation for creating landscapes in real-time. This final project will utilize the processor's ability to process data in parallel. It's also tries to overcomes the problem

where a task cannot be run in parallel but remains responsive in the eyes of the user. This is achieved by dividing the processing time that can be done on the main processor.

Testing is done by profiling this game application to determine the application's ability to process without reducing user comfort. The results obtained when using the proposed method give a processing time of 18 milliseconds to 29.87 milliseconds compared to without using it, which is 210 milliseconds to 243 milliseconds.

Key words: Processing, Game, Real-time, Procedural Content Generation

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa atas pimpinan, penyertaan, dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

OPTIMASI PROSES PEMBANGKITAN LANSKAP SECARA *REAL-TIME* MENGGUNAKAN PARALEL *PROCEDURAL CONTENT GENERATION* PADA PERMAINAN DIGITALE

Pengerjaan Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Departemen Teknik Informatika Fakultas Teknologi Elektro dan Informatika Cerdas Institut Teknologi Sepuluh Nopember.

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan penulis dapat memberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri penulis sendiri selaku pengembang.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

1. Terima kasih kepada Allah SWT, di mana penulis masih diberi kesempatan, kesehatan dan umur untuk menempuh kuliah di sini dan menjalani hidup dengan baik.
2. Ayah, Ibu, dan Kakak penulis yang selalu memberikan perhatian, dorongan, dan kasih sayang supaya lebih semangat menempuh kuliah maupun pengerjaan Tugas Akhir ini.
3. Dr. Eng. Darlis Herumurti S.Kom., M.Kom. dan juga Hadziq Fabroyir S.Kom., Ph.D. selaku Dosen Pembimbing yang telah membimbing, telah

memberikan ilmu, dan masukan kepada saya selama penyelesaian Tugas Akhir ini.

4. Seluruh karyawan Departemen Teknik Informatika ITS baik yang bekerja dikantor maupun tidak dikantor. Yang dapat membuat suasana belajar dan mengajar di Departemen ini menjadi lebih nyaman dan aman.
5. Teman-teman angkatan 2016 jurusan Informatika ITS yang telah menemani perjuangan penulis selama 4 tahun masa perkuliahan.
6. Serta pihak-pihak lain yang tidak dapat disebutkan di sini yang telah banyak membantu penulis dalam penyusunan Tugas Akhir ini.

Penulis mohon maaf apabila masih ada kekurangan pada Tugas Akhir ini. Penulis juga mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan di kemudian hari. Semoga melalui Tugas Akhir ini penulis dapat memberikan kontribusi dan manfaat yang sebaik-baiknya.

Surabaya, 2020

Firman Maulana

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan masalah, batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan buku Tugas Akhir ini.

1.1 Latar Belakang

Aplikasi permainan merupakan salah satu media hiburan yang sedang berkembang pada saat ini. Setiap tahapan permainan umumnya disusun secara manual oleh *level designer* permainan tersebut. Namun penyusunan tahapan secara manual tersebut cukup memakan waktu yang lama. Pada beberapa jenis permainan, semakin banyak tahapannya maka semakin lama pengguna akan memainkan permainan tersebut untuk mengetahui seluruh mekanisme yang ada. Hal ini membuat beberapa pengembang aplikasi permainan memutuskan untuk membuat permainan yang bertipe *open world* di mana permainan tersebut memiliki dunia yang luas. Namun pembuatan dunia yang luas ini membutuhkan waktu yang tidak sedikit. Munculnya teknologi yang lebih mutakhir memungkinkan untuk dilakukannya pembuatan aplikasi permainan dengan pendekatan yang berbeda seperti pembuatan dunia yang dinamis.

Procedural Content Generation (*PCG*) merupakan salah satu media pembuatan konten secara prosedural berdasarkan parameter yang telah ditetapkan. Pembuatan konten menggunakan algoritma ini umumnya memanfaatkan sifat acak dari algoritma *pseudo-random* yang dapat di replikasi ulang jika dievaluasi dengan parameter yang sama. Tujuan dari penggunaan *Procedural Content Generation* ini adalah untuk membuat konten yang sangat banyak dengan menggunakan waktu penyusunan yang sesedikit mungkin. Salah satu kegunaan dari algoritma ini adalah untuk membuat lanskap untuk dieksplorasi pemain. Namun pembuatan lanskap ini cenderung memakan waktu komputasi yang lama dan

umumnya dilakukan di awal saja atau diproses secara tidak sinkron.

Aplikasi permainan umumnya hanya menggunakan 1 atau 2 *thread* prosesor saja di mana *thread* ke-1 digunakan untuk mengeksekusi logika permainan dan *thread* yang ke-2 digunakan sebagai *render thread* atau proses *input* dan *output*. Hal ini menyebabkan prosesor yang memiliki banyak *thread* tidak dapat dimanfaatkan secara maksimal. Penggunaan *thread* secara berlebihan juga dapat menyebabkan degradasi performa dikarenakan *context switching*. Pembagian tugas *thread* dengan benar dapat mengurangi *context switching* sembari meningkatkan performa.

Oleh karena itu, pada tugas akhir ini akan dibuatlah sebuah implementasi algoritma *Procedural Content Generation* untuk membuat lanskap secara *real-time* dengan memanfaatkan penggunaan prosesor secara paralel. Hasil dari tugas akhir ini diharapkan dapat berkontribusi dalam industri permainan terutama pada bidang pembuatan konten secara prosedural dan juga pemanfaatan *multi-threading* dalam aplikasi permainan.

1.2 Rumusan Permasalahan

Berdasarkan hal-hal yang telah diuraikan, maka hal-hal yang menjadi permasalahan dalam tugas akhir ini adalah:

1. Bagaimana cara mengimplementasikan sistem *Procedural Content Generation* untuk membuat lanskap secara *real-time*?
2. Bagaimana cara implementasi sistem *multi-threading* pada aplikasi permainan agar dapat berkomunikasi dengan *thread* utama?
3. Bagaimana cara mengimplementasikan sistem yang mengeksplorasi lanskap dengan ukuran yang masif?

1.3 Batasan Permasalahan

Permasalahan pada Tugas Akhir ini memiliki beberapa batasan, di antaranya sebagai berikut:

1. Sistem operasi yang didukung adalah Windows 64bit.
2. Bahasa pemrograman yang digunakan adalah C#.
3. Lingkungan pengembangan yang digunakan adalah Visual Studio 2019.
4. *Game Engine* yang digunakan adalah *Unity* versi 2019.3.0f6 (64-bit),.
5. Setelah lanskap selesai dibuat maka akan disimpan untuk pemrosesan kembali ketika dibutuhkan

1.4 Tujuan Pembuatan Tugas Akhir

Tujuan dari pengerjaan tugas akhir ini adalah:

1. Mengimplementasikan sistem *Procedural Content Generation* untuk membuat lanskap secara *real-time*.
2. Mengimplementasikan sistem *multi-threading* pada aplikasi permainan agar tidak mengganggu *thread* utama.
3. Mengimplementasikan sistem yang mengeksplorasi lanskap dengan ukuran yang masif.

1.5 Manfaat Tugas Akhir

Hasil dari pengerjaan tugas akhir ini dapat berguna dalam industri permainan terutama pada bidang pembuatan konten secara prosedural dan juga pemanfaatan *multi-threading* dalam aplikasi permainan.

1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu:

1.6.1 Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi penjelasan mengenai pengimplementasian dari algoritma *procedural content generation* untuk pembuatan lanskap secara *real-time.*

1.6.2 Studi Literatur

Pada studi literatur ini, akan dipelajari sejumlah referensi yang diperlukan dalam pembuatan aplikasi yaitu mengenai bahasa pemrograman C#, *game engine Unity3D*, *procedural generation*, *coherent-noise*, *simplex noise*, *domain warping* dan *multi-threading*.

1.6.3 Implementasi Perangkat Lunak

Dalam menyelesaikan tugas akhir ini, aplikasi akan diimplementasikan menggunakan *IDE Visual Studio* dengan *game engine Unity3D* dan bahasa pemrograman C# dengan struktur yang aman untuk digunakan dengan *multi-threading*.

1.6.4 Pengujian dan Evaluasi

Tahap pengujian dan evaluasi berisi pengujian aplikasi dan evaluasi berdasarkan hasil pengujian. Pada tahap ini dilakukan pengujian dari fungsionalitas perangkat lunak, apakah sesuai dengan yang diharapkan serta diharapkan tidak terdapat *bug*. Pengujian dilakukan dengan cara mengukur performa permainan dengan cara membanding waktu yang dibutuhkan sebelum dan sesudah menggunakan metode yang akan diterapkan.

1.6.5 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat. Sistematika penulisan buku tugas akhir secara garis besar antara lain:

1. Pendahuluan
    a. Latar Belakang
    b. Rumusan Masalah
    c. Batasan Tugas Akhir
    d. Tujuan
    e. Metodologi

f. Sistematika Penulisan

2. Tinjauan Pustaka
3. Analisis dan Perancangan Sistem
4. Implementasi
5. Pengujian dan Evaluasi
6. Kesimpulan dan Saran
7. Daftar Pustaka.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini merupakan laporan secara lengkap mengenai Tugas Akhir yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut. Sistematika penulisan buku Tugas Akhir secara garis besar antara lain:

Bab I Pendahuluan

Bab ini berisi penjelasan latar belakang, rumusan masalah, batasan masalah dan tujuan pembuatan Tugas Akhir. Selain itu, metodologi pengerjaan dan sistematika penulisan laporan Tugas Akhir juga terdapat di dalamnya.

Bab II Tinjauan Pustaka

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Perancangan Sistem

Bab ini berisi penjelasan tentang rancangan dari sistem yang akan dibangun.

Bab IV Implementasi

Bab ini berisi penjelasan implementasi dari rancangan yang telah dibuat pada bab sebelumnya.

Implementasi disajikan dalam bentuk *pseudocode* disertai dengan penjelasannya.

Bab V Uji coba dan Evaluasi

Bab ini berisi penjelasan mengenai data hasil percobaan dan pembahasan mengenai hasil percobaan yang telah dilakukan.

Bab VI Kesimpulan dan Saran

Bab ini merupakan bab terakhir yang menjelaskan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak kedepannya.

BAB II
TINJAUAN PUSTAKA

Pada bagian ini akan dibahas mengenai teori-teori yang menjadi dasar dari penyelesaian tugas akhir penulis. Pembahasan bertujuan untuk memberikan gambaran secara umum terhadap program yang akan dibuat dan berguna sebagai penunjang dalam pengembangan tugas akhir.

2.1 Bahasa Pemrograman C#

C# (dibaca: *C sharp*) merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh *Microsoft* sebagai bagian dari inisiatif kerangka *.NET Framework*. Bahasa pemrograman ini dibuat berbasiskan bahasa *C++* yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti *Java*, *Delphi*, *Visual Basic*, dan lain-lain) dengan beberapa penyederhanaan.

2.2 Game Engine Unity

Unity adalah salah satu *game engine* lintas *platform* yang dikembangkan oleh *Unity Technologies* dimana pertama kali diumumkan dan dirilis ke publik pada juni 2005 di konferensi *Apple Inc* sebagai *game engine* eksklusif pada sistem operasi *Mac OS X*. *Game engine* ini dapat digunakan untuk membuat aplikasi permainan 2-dimensi, 3-dimensi, realitas virtual, realitas teraugmentasi dan juga simulasi. *Game engine* ini juga telah diadopsi di luar aplikasi permainan yaitu film, otomotif, arsitektur, rekayasa, dan konstruksi.

2.3 Procedural Generation

Procedural Generation adalah metode untuk menciptakan data secara algoritmik dan bukan secara manual, biasanya melalui

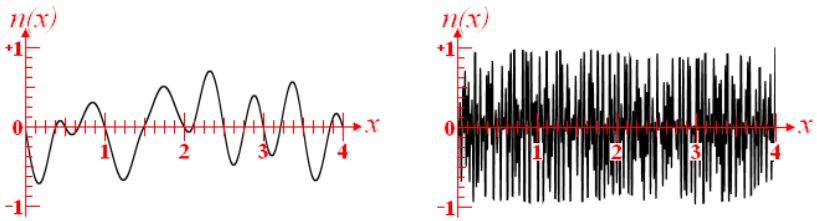
kombinasi aset dan algoritma yang dihasilkan manusia ditambah dengan keacakan yang dihasilkan komputer dan kekuatan pemrosesan. Dalam grafik komputer, biasanya digunakan untuk membuat tekstur dan model 3D. Dalam aplikasi permainan, ini digunakan untuk secara otomatis membuat konten dalam jumlah besar dalam sebuah permainan. Keuntungan dari pembuatan prosedural termasuk ukuran berkas yang lebih kecil dan jumlah konten yang lebih besar.

2.4 Coherent Noise

Coherent noise adalah salah satu tipe dari pseudo-*random noise* yang teratur. *Coherent noise* didapatkan dari fungsi *coherent-noise* di mana fungsi tersebut memiliki 3 properti utama yaitu :

- Memberikan masukan yang sama akan menghasilkan keluaran yang sama.
- Perubahan yang sedikit pada masukan akan menghasilkan sedikit perubahan pada keluaran.
- Perubahan yang banyak pada masukan akan menghasilkan perubahan yang terlihat acak pada keluaran.

Sebuah *coherent-noise* dengan masukan berdimensi-n akan menghasilkan keluaran bernilai skalar. Gambar 2.1 menunjukkan perbandingan antara *coherent-noise* dengan *random-noise*.



Gambar 2.1 Perbedaan *coherent-noise* (kiri) dan *random-noise* (kanan).

2.5 Simplex Noise

Simplex Noise adalah metode untuk membangun fungsi noise dengan dimensi-n yang sebanding dengan *Perlin Noise* (*noise* "klasik") tetapi dengan artefak arah yang lebih sedikit dan mengurangi biaya komputasi pada dimensi yang lebih tinggi. Ken Perlin merancang algoritma ini pada tahun 2001 untuk mengatasi keterbatasan fungsi *noise* klasiknya, terutama dalam dimensi yang lebih tinggi.

Agar fungsi *noise* dapat diulang dan selalu menghasilkan nilai yang sama untuk titik *input* yang diberikan, gradien harus pseudo-acak, dan bukan benar-benar acak. Gradien tersebut perlu memiliki variasi yang cukup untuk menyembunyikan bahwa fungsi tersebut tidak benar-benar acak, tetapi terlalu banyak variasi akan menyebabkan perilaku yang tidak dapat diprediksi untuk fungsi *noise*. Pilihan yang baik untuk 2D dan lebih tinggi adalah untuk memilih gradien panjang unit tetapi arah yang berbeda. Untuk fungsi 2 dimensi, 8 atau 16 gradien yang didistribusikan di sekitar lingkaran unit adalah pilihan yang baik. Untuk fungsi 3 dimensi, serangkaian gradien yang direkomendasikan Ken Perlin adalah titik tengah dari masing-masing 12 tepi kubus yang berpusat pada titik asal.

Pada tugas akhir ini akan digunakan implementasi *Simplex Noise* yang bernama *Open Simplex* dengan pustaka bernama *FastNoise*.

2.6 Perlin Noise

Perlin noise adalah jenis *gradient noise* yang dikembangkan oleh Ken Perlin pada tahun 1983 sebagai akibat dari frustrasinya dengan tampilan yang seperti mesin dari citra yang dihasilkan komputer pada saat itu. Dia secara resmi menggambarkan temuannya dalam makalah *SIGGRAPH* pada tahun 1985 yang disebut "*An image Synthesizer*". Perlin

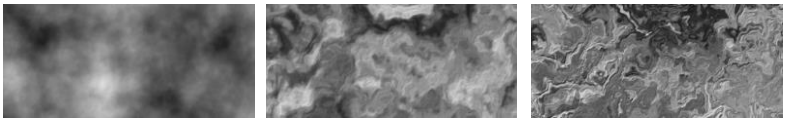
dianugerahi *Academy Award* untuk *Technical Achievement* karena menciptakan algoritma ini.

Perlin noise adalah tekstur prosedural primitif, sejenis *gradient noise* yang digunakan oleh seniman efek visual untuk meningkatkan tampilan realisme dalam grafik komputer. Fungsi ini memiliki tampilan pseudo-acak, namun semua detail visualnya berukuran sama. Properti ini memungkinkannya mudah dikendalikan; beberapa salinan skala *perlin noise* dapat dimasukkan ke dalam ekspresi matematis untuk menciptakan berbagai macam tekstur prosedural. Tekstur sintetis menggunakan *perlin noise* sering digunakan dalam grafika komputer untuk membuat elemen visual yang dihasilkan komputer - seperti permukaan objek, api, asap, atau awan - tampak lebih alami, dengan meniru tampilan acak tekstur yang terkendali di alam.

Perlin noise juga sering digunakan untuk menghasilkan tekstur ketika memori sangat terbatas, seperti dalam demo. Penggantinya, seperti *fractal noise* dan *simplex noise*, telah menjadi hampir di mana-mana di unit pemrosesan grafis baik untuk grafik *real-time* maupun untuk tekstur prosedural *non-real-time* di semua jenis grafik komputer.

2.7 Domain Warping

Domain warping adalah teknik yang umum digunakan untuk membuat tekstur dan obyek geometris pada komputer grafis. Teknik ini memberikan efek deformasi pada sebuah tekstur atau obyek geometris jika dan hanya jika fungsi evaluasi yang digunakan merupakan fungsi yang menyatakan suatu ruang. Teknik ini pertama kali digunakan oleh Ken Perlin pada tahun 1984 untuk membuat tekstur marmer prosedural pertama. Gambar 2.2 menunjukkan hasil dari penggunaan *domain warping* untuk memodifikasi tekstur yang ada.



Gambar 2.2 Hasil dari iterasi *domain warping* sebanyak 1, 2, dan 3 kali.

2.8 Multithreading

Multithreading biasanya ditemukan di sistem operasi *multitasking*. *Multithreading* adalah model pemrograman dan eksekusi yang luas yang memungkinkan banyak *thread* ada dalam konteks satu proses. *Thread* ini membagikan sumber daya proses, tetapi dapat dieksekusi secara independen. Model pemrograman ber-*thread* menyediakan pengembang dengan abstraksi yang berguna dari eksekusi paralel. *Multithreading* juga dapat diterapkan pada satu proses untuk memungkinkan eksekusi paralel pada sistem *multiprocessing*.

[Halaman ini sengaja dikosongkan]

BAB III
PERANCANGAN SISTEM

Pada bagian ini akan dijelaskan analisis dan desain sistem yang digunakan untuk menyelesaikan permasalahan pada Tugas Akhir ini.

3.1 Deskripsi Umum Sistem

Sistem ini merupakan sistem yang berbasis pada eksplorasi di mana pemain dapat mengeksplorasi dunia yang dibuat secara prosedural untuk memberikan kesan dunia yang masif. Fokus pada tugas akhir ini adalah bagaimana cara membuat dunia yang terlihat masif tanpa mengorbankan tingkat kelayakan permainan ini dengan menggunakan kemampuan pemrosesan secara paralel.

Pada permainan ini akan disediakan sebuah lanskap yang dibuat secara prosedural berdasarkan jangkauan pemain saat ini. Dikarenakan sifat prosedural yang dihasilkan pemain dapat terus mengeksplorasi ke arah mana pun yang mereka inginkan. Contoh dari aplikasi permainan yang sudah menerapkan sistem pembuatan secara prosedural ditunjukkan oleh gambar 3.1 yaitu *No Man's Sky* dan juga gambar 3.2 yaitu *Minecraft*.

Gambar 3.1 Tangkapan layar dari permainan No Man's Sky.

Gambar 3.2 Tangkapan layar dari permainan *Minecraft*.

3.2 Desain Permainan

Pemain dapat mengeksplorasi lanskap yang ada secara *real-time*. Pembuatan lanskap ditentukan berdasarkan posisi pemain saat ini. Lanskap yang dibuat memiliki beberapa fitur tersendiri seperti gunung, lembah, dan danau. Pada setiap fitur terdapat beberapa jenis vegetasi. Untuk menyimulasikan ragam vegetasi maka dibuatlah beberapa *prefab*. *Prefab – prefab* ini dibentuk berbasiskan dodekahedron untuk memberikan kesan *low poly* pada permainan ini.

3.3 Desain Sistem

Sistem terdiri dari kumpulan fungsi - fungsi yang digunakan untuk melakukan pembuatan lanskap secara *real-time*.

3.3.1 Desain Arsitektur Sistem

Seluruh lanskap dibuat dalam bentuk *chunk* atau petak dan dibuat ketika dibutuhkan saja untuk menghemat penggunaan sumber daya yang ada. Jika pemain berpindah dalam radius tertentu maka posisi terbaru akan diperbaharui dan mulai diproses. Ketika sudah pernah diproses maka sistem akan otomatis

mengambil yang sudah diproses. Hal ini ditujukan untuk mengurangi beban pemrosesan ke depannya.

Tahap awal pemrosesan adalah dengan memberikan permintaan pemrosesan kepada sistem. Sistem akan mengkoordinasikan permintaan untuk mulai membuat data hingga data digabungkan kembali. Sebagian proses yang dapat diparalelkan akan dijalankan secara paralel dan sebagian proses yang tidak dapat diparalelkan akan diantrekan di *thread* utama dan diberikan waktu pemrosesan.

Setelah permintaan pemrosesan diterima, sistem akan mulai membuat *noise map* dalam bentuk *array* 2 dimensi. Pembuatan *noise map* ini memanfaatkan sebuah *noise pipeline* untuk membuat *noise map* yang menarik. Pada proses ini terjadi pemrosesan secara paralel dikarenakan tidak ada dependensi dengan proses yang lain.

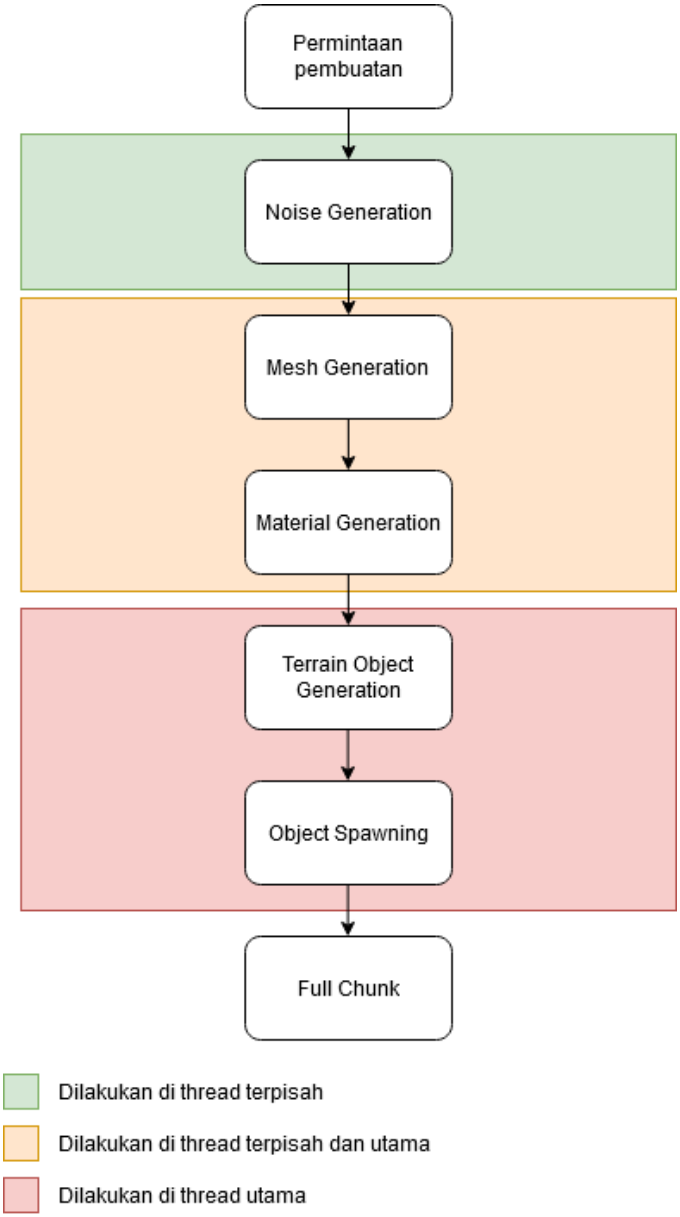
Setelah selesai membuat *noise map*, maka dibuatlah tekstur dari *noise map* tersebut. *Noise map* ini kemudian digunakan untuk membuat *mesh* atau objek yang digunakan untuk merepresentasikan sebuah lanskap kecil. Pada tahap ini terjadi 4 sub proses yaitu membuat *vertex*, komputasi *UV map*, membuat *triangle* dari *vertex* tadi, dan menggabungkan hasilnya menjadi *mesh* utuh. Untuk 3 proses pertama dapat dilakukan secara paralel namun untuk proses kombinasi hanya dapat dilakukan di *thread* utama.

Noise map yang masuk akan digunakan untuk membuat tekstur warna yang menunjukkan ketinggian dari suatu titik di *noise map* tersebut. Warna tertentu menandakan fitur tertentu. Pada proses ini terjadi 2 macam metode pemrosesan. Pemrosesan pembuatan *pixel – pixel* yang ada dilakukan secara paralel. Namun proses penyatuan kumpulan *pixel* tadi untuk menjadi sebuah tekstur dan material dilakukan di *thread* utama. Hal ini dikarenakan pembuatan tekstur dan material dari *pixel* membutuhkan pemanggilan fungsi yang membutuhkan akses dari *API* yang hanya bisa diakses di *thread* utama.

Setelah semua proses selesai maka, semuanya akan digabungkan menjadi satu obyek utuh, Setelah menjadi obyek utuh, maka sistem akan mulai mengkomputasi posisi obyek dan juga *collider* yang dibutuhkan dan meletakkan obyek di *game*.

Setelah *chunk* terbuat maka proses selanjutnya adalah dengan mulai meletakkan properti yang dibutuhkan. Contoh properti yang akan diletakkan adalah pohon dan juga rumput.

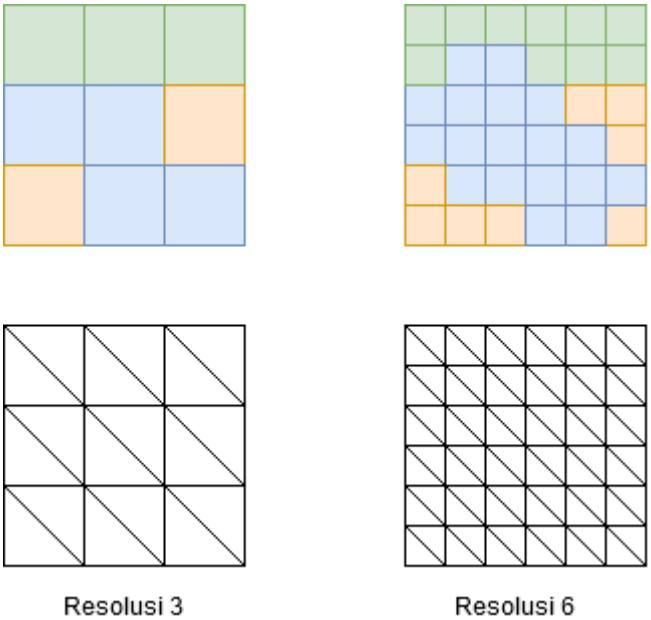
Gambar 3.3 menjelaskan tentang gambaran keseluruhan dari proses pembuatan *chunk*.


Permintaan pembuatan
Noise Generation
Mesh Generation
Material Generation
Terrain Object Generation
Object Spawning
Full Chunk
Dilakukan di thread terpisah
Dilakukan di thread terpisah dan utama
Dilakukan di thread utama

Gambar 3.3 Proses pembuatan sebuah petak.

3.3.2 Desain Chunk Generation

Setiap *chunk* atau petak yang dibuat akan diproses ketika dibutuhkan saja. Hal ini ditujukan untuk menghemat sumber daya yang ada. *Chunk* akan terbuat ketika pemain berpindah tempat dari satu petak ke petak lainnya. Hal ini dilakukan dengan cara memantau posisi pemain saat ini. Jika pemain berpindah tempat maka akan dikalkulasikan perubahan yang harus diterapkan. Perubahan ini meliputi *level of detail* dan obyek yang tersedia. Setiap petak memiliki resolusi dan ukuran tersendiri. Semakin tinggi resolusi dari sebuah petak maka semakin detail fitur – fitur dari petak tersebut. Ukuran petak hanya menentukan ukuran obyek dalam permainan. Jika kenaikan ukuran tidak diimbangi dengan kenaikan resolusi maka hasil dari petak tersebut akan terlihat seperti *low poly*. Gambar 3.4 mengilustrasikan dampak dari perbedaan resolusi.



Gambar 3.4 Ilustrasi dari perbedaan resolusi.

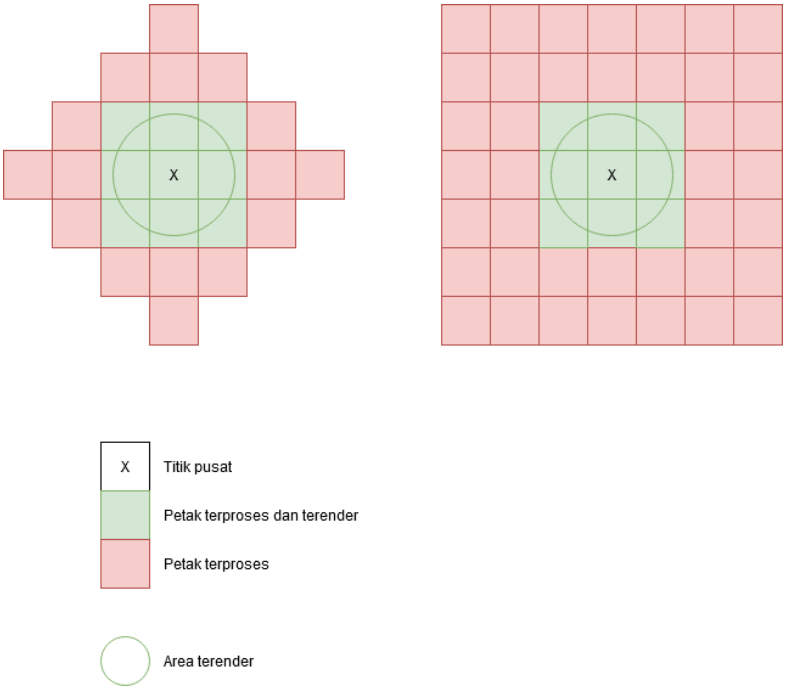
3.3.3 Desain Level Of Detail

Untuk menghemat sumber daya yang ada maka pembuatan petak tidak dilakukan pada resolusi tertinggi. Penentuan resolusi dilakukan dengan memperhatikan jarak dari petak yang akan diproses. Semakin jauh posisi petak yang akan diproses terhadap pemain, maka semakin rendah resolusinya. Penurunan resolusi untuk setiap tahap kejauhan sebesar setengah dari tahap sebelumnya. Setiap tahap resolusi bisa saja memiliki ukuran yang berbeda – beda. Gambar 3.5 menjelaskan tentang perbedaan ukuran setiap tahap. Gambar 3.6 mengilustrasikan setiap tahap resolusi dalam bentuk belah ketupat.

Gambar 3.5 Ilustrasi perbedaan resolusi pada setiap tahap.

Gambar 3.6 Ilustrasi setiap tahap resolusi berdasarkan bentuk belah ketupat.

Struktur petak yang akan dibuat harus efektif. Dengan mempertimbangkan bahwa peletakan petak yang paling efisien berupa lingkaran maka petak yang akan dibuat berbentuk belah ketupat. Hal ini dikarenakan bentuk ini merupakan bentuk yang paling dekat dengan lingkaran. Gambar 3.7 menjelaskan perbedaan jumlah petak yang terproses dan ter-*render*.

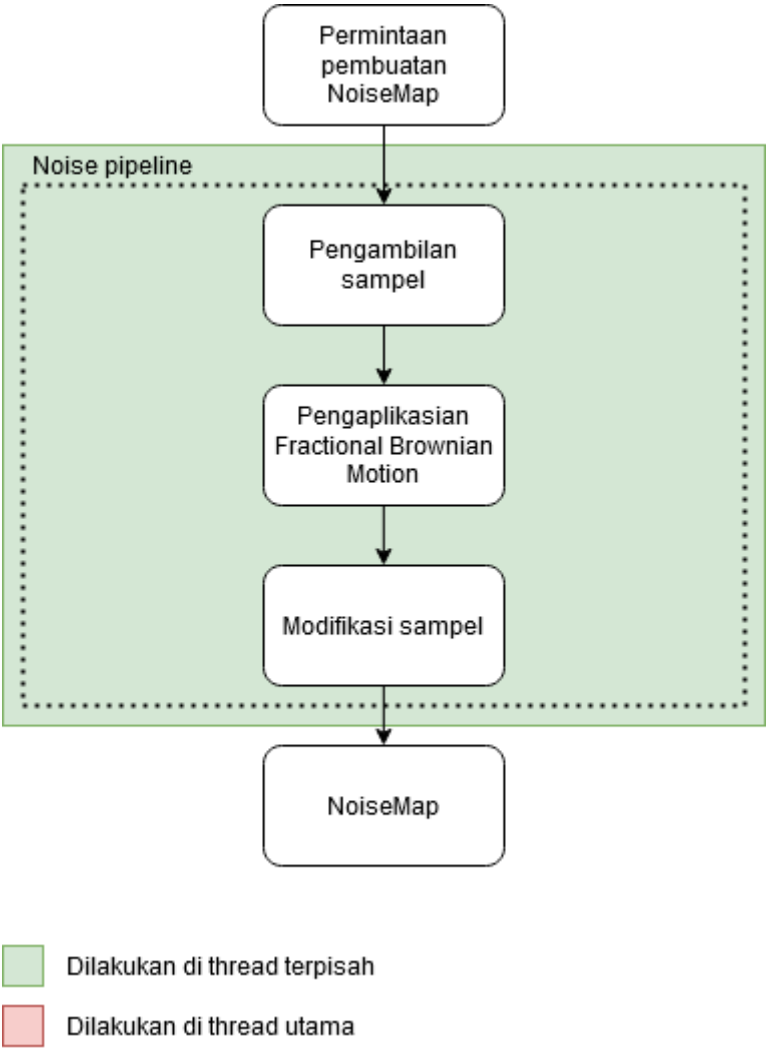


Gambar 3.7 Ilustrasi dari area yang terproses dan ter-*render*.

3.3.4 Desain Noise Generation

Pada proses ini terjadi pembuatan *noise map*. Keluaran dari proses ini adalah sebuah *noise map* berbentuk *array* 2 dimensi. Pengambilan sampel di tiap titik didasarkan pada posisi yang ingin disampel dan dilakukan skalasi dengan nilai skala yang diinginkan. Proses ini akan melakukan *domain warping* dengan menggunakan *fractional brownian motion* untuk dapat memberikan kesan terdistorsi pada keluaran. *Fractional brownian motion* akan dilakukan menggunakan pengambil sampel yang sama dengan menambahkan sebagian nilai dari komputasi *fractional brownian motion* pada sebelumnya. Komputasi ini diulang sebanyak jumlah iterasi yang diinginkan. Untuk proses pengambilan sampel *noise* di titik tertentu dilakukan dengan menggunakan *simplex noise*

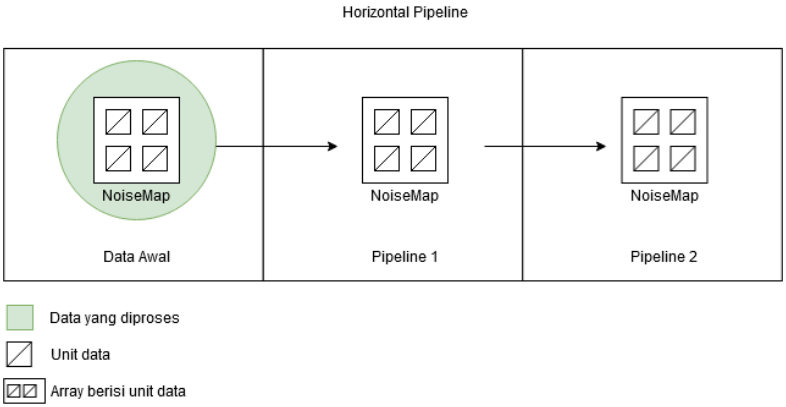
ataupun *perlin noise* untuk memberikan sebuah *coherent noise*. Proses komputasi sampel dilakukan oleh pustaka pihak ketiga yang akan digunakan di mana untuk *simplex noise* akan di komputasi oleh pustaka *OpenSimplex Noise* dan untuk *perlin noise* akan menggunakan implementasi dari *Unity*. Setelah sebuah seluruh sampel berhasil di komputasi maka sampel tersebut akan dilarikan ke *pipeline* yang ada untuk dimodifikasi. Proses modifikasi melalui *pipeline* bisa saja melibatkan pengambilan sampel ulang berdasarkan nilai sampel saat ini. Terdapat 2 jenis *pipeline* untuk digunakan membuat lanskap yang ada, yaitu *horizontal pipeline* dan *vertical pipeline*. Istilah *pipeline* dipilih dikarenakan bentuk pemrosesan yang dilakukan. Semua komputasi yang dilakukan pada proses ini dilakukan di *thread* yang terpisah. Gambar 3.8 menunjukkan sub proses yang terjadi.



Gambar 3.8 Proses pembuatan *noise map*.

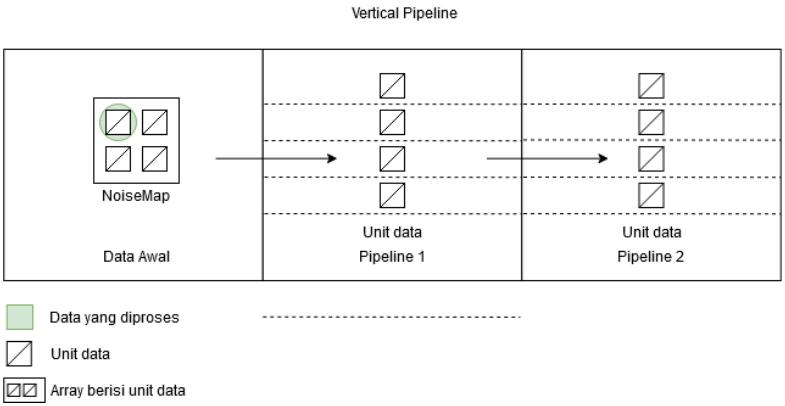
Pada *horizontal pipeline* data masuk dan di proses secara keseluruhan. Dikarenakan pemrosesan secara keseluruhan ini maka dapat dilakukan manipulasi data terhadap data tetangganya

maupun memodifikasi data secara keseluruhan. Akan tetapi *pipeline* ini akan berjalan sangat lambat dikarenakan proses iterasi jika pemrosesan dilakukan terhadap seluruh data yang ada dan bukan data tertentu saja. *Pipeline* ini cocok digunakan untuk memproses data – data kecil namun membutuhkan data lainnya yang berada pada *pipeline* yang sama. Contoh operasi yang dapat dilakukan pada *pipeline* ini adalah untuk mereratakan area di sekitarnya. Gambar 3.9 menunjukkan ilustrasi dari *horizontal pipeline*.



Gambar 3.9 Ilustrasi dari *horizontal pipleine*.

Pada *vertical pipeline* data masuk dan diproses satu persatu. Hal ini digunakan untuk mentransformasi seluruh data yang ada dari kondisi A ke kondisi B. Dikarenakan data diproses satu persatu maka data lainnya menjadi tidak dapat diakses. *Pipeline* ini cocok digunakan untuk memanipulasi total seluruh data yang masuk. Contoh operasi yang dapat dilakukan *pipeline* ini adalah multiplikasi dan *clamping*. Gambar 3.10 menunjukkan ilustrasi *vertical pipeline*.

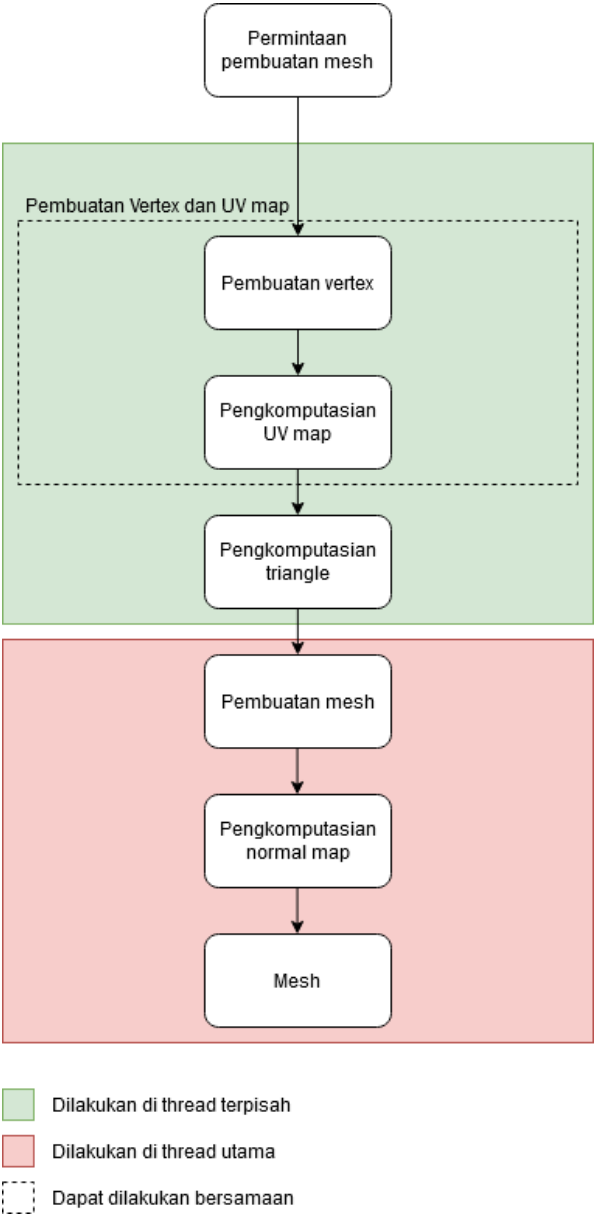


Gambar 3.10 Ilustrasi dari *vertical pipeline*.

3.3.5 Desain Mesh Generation

Pada proses ini terjadi pembuatan *mesh* yang akan dibuat menjadi sebuah representasi 3 dimensi. Dalam proses ini terjadi 4 sub proses yaitu membuat *vertex*, membuat *triangle* dari *vertex* tadi, menkomputasi *UV map*, dan menggabungkan hasilnya menjadi *mesh* utuh.

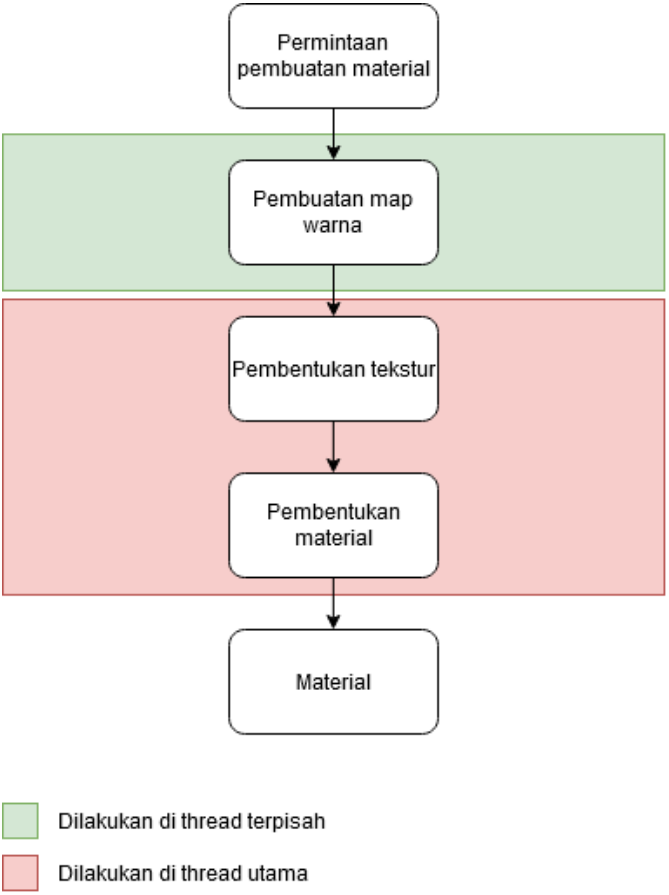
Pada sub proses pertama dibuatlah *noise* map yang ada menjadi kumpulan *vertex*. Kumpulan *vertex* ini disusun berdasarkan posisi *noise* map saat ini. Setiap *vertex* yang telah dibentuk kemudian akan diubah menjadi sebuah *triangle*. *Triangle* ini di komputasi berdasarkan urutan tertentu berdasarkan masukan dari *vertex* yang ada. Untuk membuat sebuah *face* maka dibutuhkanlah 3 *vertex* untuk dijadikan sebuah *triangle*. Untuk meletakkan tekstur pada obyek maka dibutuhkannya komputasi dari UV map. Komputasi UV map dijalankan bersamaan dengan komputasi *vertex* untuk menghemat waktu dan dikarenakan mesh yang akan dibentuk berupa *plane*. Setelah itu seluruh *vertex*, *triangle* dan *UV map* tadi dibentuk menjadi sebuah *mesh*. *Mesh* tersebut di komputasi ulang normalnya untuk memberikan kesan yang halus. Gambar 3.11 menjelaskan sub proses yang terjadi pada saat pembuatan *mesh*.


Permintaan pembuatan mesh
Pembuatan Vertex dan UV map
Pembuatan vertex
Pengkomputasian UV map
Pengkomputasian triangle
Pembuatan mesh
Pengkomputasian normal map
Mesh
Dilakukan di thread terpisah
Dilakukan di thread utama
Dapat dilakukan bersamaan

Gambar 3.11 Sub proses dari *mesh generation*.

3.3.6 Desain Material Generation

Pada proses ini terjadi pembuatan material untuk digunakan pada *terrain* yang akan dibuat. Proses ini membutuhkan *noise map* yang telah dibuat pada proses sebelumnya untuk digunakan sebagai referensi penentuan warna yang akan digunakan. Proses pembuatan tekstur ini didasarkan pada nilai *noise map* pada titik tertentu. Setelah warna sudah ditentukan maka dibuatlah tekstur dari kumpulan warna yang ada. Setelah tekstur terbentuk kemudian dibuatlah material untuk meletakkan tekstur yang ada. Gambar 3.12 menjelaskan tentang proses yang terjadi pada pembuatan material.

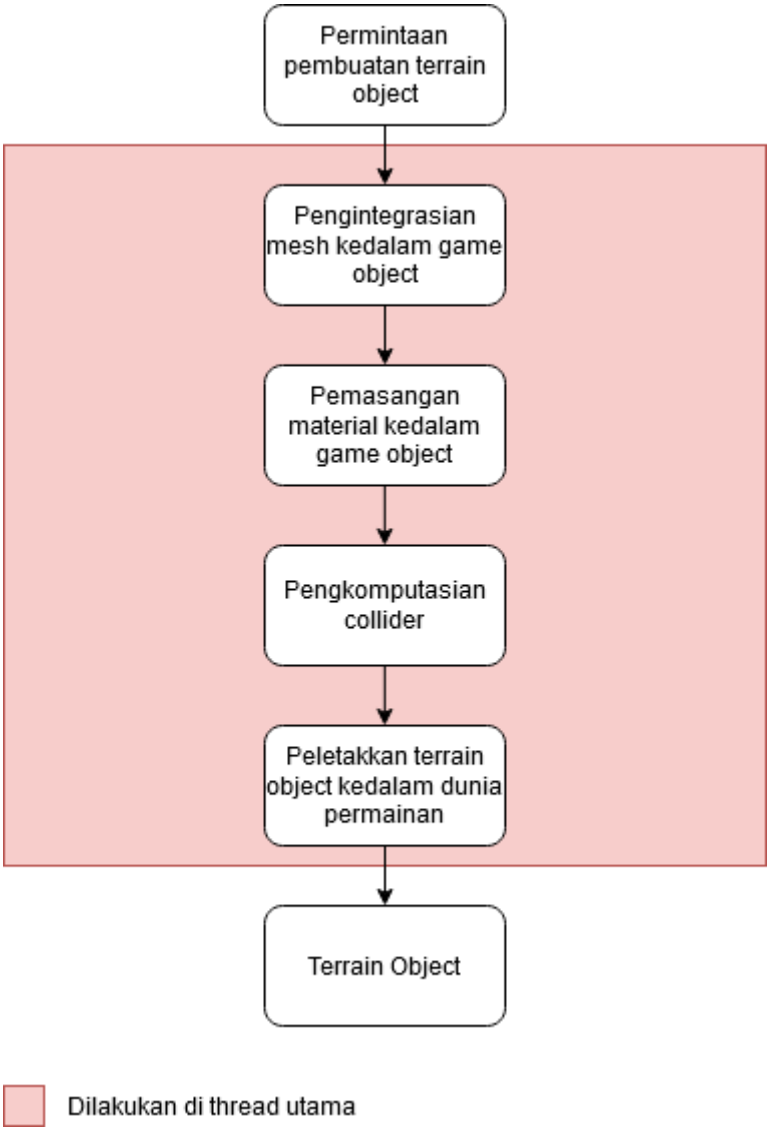


Gambar 3.12 Ilustrasi proses pembuatan material.

3.3.7 Desain Terrain Object Generation

Pada proses ini dilakukannya pembuatan *terrain object* berdasarkan proses – proses sebelumnya. Proses ini membutuhkan *mesh*, dan material dari proses sebelumnya. Proses ini akan meletakkan *mesh* kedalah *game object* yang dibuat lalu

memberikannya material untuk *mesh* tersebut. Setelah itu dilakukanlah komputasi untuk mendapatkan *collider* dari *game object* ini. Setelah selesai maka terrain object akan diletakkan ke dunia permainan. Gambar 3.13 menunjukkan proses pembuatan dari *terrain object*.



Gambar 3.13 Ilustrasi proses pembuatan *terrain object*.

3.3.8 Desain Object Spawning

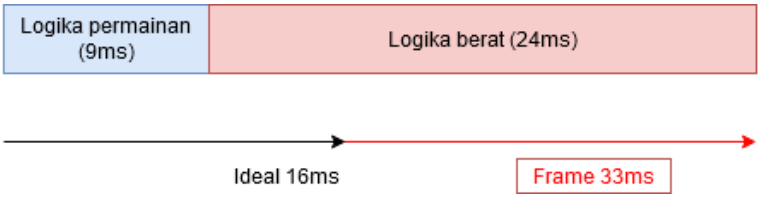
Pada proses ini dimulailah dilakukannya peletakan obyek - obyek yang harus diletekkan pada dunia permainan. Proses peletakan obyek harus konsisten jika dievaluasi dari titik yang sama. Untuk menjaga konsistensi ini maka, obyek yang akan di letakkan haruslah berbasis pada nilai yang tidak berubah. Maka dari itu komputasi obyek yang akan diletakkan menggunakan nilai dari *noise* di titik tersebut. Untuk mendapatkan kesan yang agak acak maka digunakanlah nilai di belakang koma yang dihasilkan oleh *noise* di titik tersebut. Hal ini dikarenakan nilai tersebut selalu sama jika di evaluasi dititik tertentu. Proses peletakan juga dilakukan berdasarkan kriteria tertentu seperti kemungkinan untuk diletakkan dan juga ketinggian. Misal saja, pohon bersalju tidak mungkin ada di daerah dengan nilai *noise* di bawah 0,7. Hal ini ditujukan untuk memberikan kesan seragam terhadap pengguna. Obyek – obyek yang diletakkan juga tidak dibuat secara terus menerus melainkan akan disimpan untuk digunakan kembali ketika sudah tidak lagi diperlukan. Hal ini ditujukan untuk menghemat waktu pemrosesan yang dibutuhkan untuk membuat obyek. Objek juga hanya akan diletakkan pada petak yang berada di tahap awal. Hal ini dikarenakan pemain kemungkinan besar hanya akan berinteraksi dengan obyek yang dekat dengan mereka. Gambar 3.14 mengilustrasikan contoh persebaran obyek berupa batu dan pohon pada area tertentu di mana area menengah memiliki lebih banyak pohon dan area air tidak memiliki pohon sama sekali.

Gambar 3.14 Ilustrasi distribusi obyek pada petak.

3.3.9 Desain Task Scheduling

Seluruh proses yang terjadi memiliki cara pemrosesan tersendiri. Hal ini diperlukan karena tidak semua proses dapat dijalankan di *thread* yang terpisah. Semakin banyak proses yang bisa dijalankan di *thread* terpisah maka akan semakin baik. Hal ini dikarenakan *thread* utama dapat memproses logika permainan tanpa adanya hambatan yang dapat memperlambat pemrosesan. Jenis pemrosesan yang dapat dilakukan terbagi menjadi 3 yaitu: pemrosesan penuh di *thread* utama, pemrosesan secara paralel, dan pemrosesan berwaktu di *thread* utama.

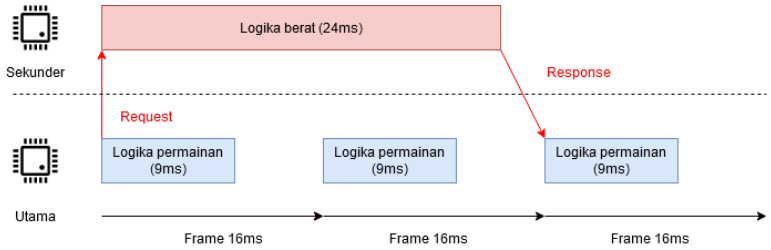
Untuk pemrosesan penuh di *thread* utama hanya dilakukan jika logika yang di proses merupakan logika yang tidak dapat ditunda waktu pemrosesannya. Hal ini dikarenakan adanya kebutuhan untuk memproses logika yang memiliki dampak langsung dari aktivitas pemain dan tidak dapat ditunda. Contoh dari ini adalah mengetahui masukan dari pemain dan memberikan umpan balik berupa pembaruan posisi. Jika hal ini ditunda pemrosesannya maka dapat membuat pemain bingung dan mengira permainan ini tidak responsif. Gambar 3.15 mengilustrasikan tentang logika berat yang di proses penuh di *thread* utama.



Gambar 3.15 Ilustrasi pemrosesan logika permainan secara umum.

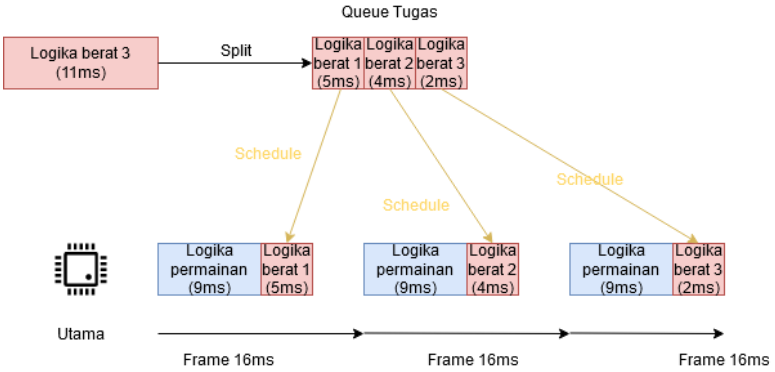
Untuk pemrosesan secara paralel, pemrosesan jenis ini dilakukan di *thread* terpisah agar tidak mengganggu *thread* utama. Pemrosesan jenis ini hanya dapat dilakukan untuk tugas yang tidak memerlukan interaksi secara langsung dengan pemain maupun dengan *game engine*. Pemrosesan jenis ini juga cocok digunakan untuk memproses sesuatu yang sangat berat dan kemungkinan dapat memakan waktu yang lama. Untuk pemrosesan jenis ini juga

tidak dapat dilakukan secara langsung. Untuk mendapatkan hasil yang maksimal dan tidak mengganggu *thread* utama maka perlu dilakukannya pembatasan jumlah *thread* yang boleh digunakan untuk memproses secara paralel. Tidak hanya itu, penggunaan *thread* secara berlebihan dapat menurunkan performa pemrosesan dikarenakan *thread* yang sedang memproses harus mengganti proses ke proses lain jika jumlah *thread* yang tersedia lebih sedikit dari *thread* yang diminta. Hal ini disebut juga dengan *context switching*. Waktu pemrosesan dapat terbuang hanya karena sering terjadinya *context switching*. Untuk mengatasi masalah – masalah tersbut maka perlu dilakukannya *thread polling*. Hal ini dengan cara mengantrekan logika yang akan diproses dan membatasi jumlah *thread* yang boleh mengatasi tugas yang sedang diantrekan tersebut. Agar tidak mengganggu *thread* utama maka jumlah *thread* yang boleh digunakan adalah sebanyak N – 2, di mana N merupakan jumlah *thread* yang tersedia untuk pemrosesan. Namun jika platform pemrosesan tidak mempunya lebih dari 2 *thread* maka jumlah *thread* yang boleh digunakan hanyalah 1. Hal ini dikarenakan *thread* utama tidak boleh diberikan beban selain logika permainan dikarenakan dapat memberikan persepsi kepada pengguna bahwa permainan ini di optimasi dengan baik. Setelah logika yang diminta selesai di proses maka, hasilnya akan di antrekan di *thread* utama untuk digabungkan dengan hasil yang lain. Gambar 3.16 mengilustrasikan kondisi ketika mengerjakan logika berat dengan pemrosesan secara paralel.



Gambar 3.16 Ilustrasi distribusi tugas ke *thread* sekunder.

Untuk pemrosesan berwaktu di *thread* utama, dilakukan dengan cara memberikan waktu pemrosesan yang dapat dilakukan oleh sebuah logika dalam tiap *frame*. Proses jenis ini merupakan salah satu cara untuk dapat menghindari pemrosesan yang berat di utama agar tidak mengganggu jalannya permainan. Proses ini cocok digunakan untuk logika yang tidak dapat dijalankan secara paralel namun merupakan salah satu logika yang berat. Untuk menggunakan proses ini maka logika permainan harus dibuat sekecil mungkin untuk menghindari pemrosesan melebihi batas waktu. Proses akan di antrekan di *thread* utama dan akan dieksekusi tiap *frame*. Setiap *frame* akan dialokasikan waktu pemrosesan sebesar yang diinginkan. Jika waktu pemrosesan sudah habis namun belum semua logika diselesaikan maka pemrosesan harus menunggu *frame* berikutnya. Contoh dari logika yang dapat memanfaatkan pemrosesan jenis ini adalah untuk komputasi *collider*. Hal ini dapat dilakukan karena *Unity* dapat melakukan komputasi terhadap *collider* secara otomatis namun hanya dapat dilakukan di *thread* utama. Gambar 3.17 menjelaskan tentang pemrosesan logika berat dengan memanfaatkan pembatasan waktu eksekusi di *thread* utama.



Gambar 3.17 Ilustrasi reduksi tugas dan pengeksekusian antar *frame*.

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Bab ini membahas implementasi dari perancangan sistem sesuai dengan perancangan yang telah dibuat. Bahasa pemrograman yang digunakan untuk implementasi sistem adalah Bahasa pemrograman C#.

4.1 Lingkungan Implementasi

Lingkungan implementasi sistem yang digunakan untuk mengembangkan tugas akhir ini memiliki spesifikasi perangkat keras dan perangkat lunak yang ditunjukkan oleh Tabel 4.1.

Tabel 4.1 Spesifikasi Lingkungan Implementasi.

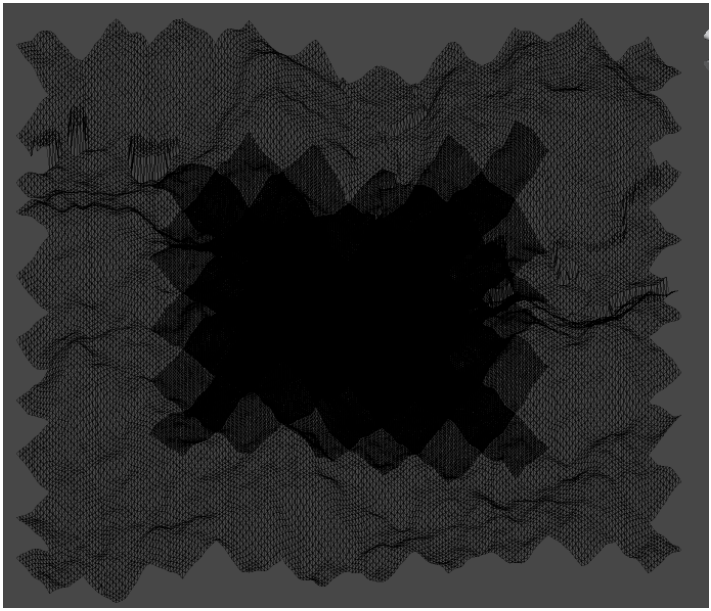
Perangkat	Spesifikasi
Perangkat Keras	• Perangkat pengembangan sistem: *Intel Core* i7-7700HQ 2.8GHz, *NVIDIA GeForce* GTX 1050TI 4GB, RAM 16GB
Perangkat Lunak	• Sistem operasi: *Microsoft Windows* 10 *Home* 64-bit 1909 • Perangkat pengembang: *Unity* 2019.3.0f6 (64-bit), *Visual Studio Community* 2019 16.4.5

4.2 Implementasi Manager

Kelas ini bertugas untuk mengatur seluruh komponen yang berada pada dunia permainan. Kelas ini memiliki banyak tugas yang di antaranya adalah untuk menginisialisasi komponen lainnya, menginisialisasi parameter pemrosesan, memantau perubahan kondisi pemain, menginstruksi permintaan pembaruan kondisi, dan mengintegrasikan *chunk* yang sudah dibuat ke dalam dunia permainan.

Proses inisialisasi komponen terjadi pada fungsi *Awake* dan *Start*. Di sini seluruh komponen diinisialisasi ataupun dicari keberadaannya dalam dunia permainan. Proses pencarian komponen eksternal menggunakan posisi relatif dari obyek ini ke komponen yang dicari. Komponen lainnya yang tidak memerlukan akses eksternal dapat diinstansiasi secara langsung.

Proses inisialisasi parameter pemrosesan dilakukan pada fungsi *Awake* dan *Start* di mana terjadi komputasi parameter yang akan digunakan untuk menentukan pembuatan dari *chunk* yang akan dibuat beserta *level of detail* dari *chunk* tersebut. Gambar 4.1 menunjukkan pengimplementasian *level of detail* di mana *chunk* terdalam memiliki tingkat kedetailan tertinggi dan *chunk* terluar memiliki tingkat kedetailan paling rendah.

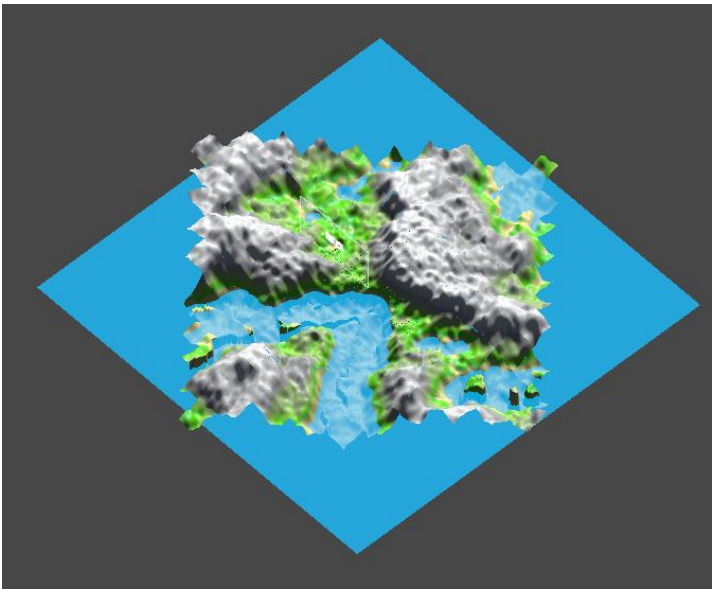


Gambar 4.1 Menunjukkan tingkat kedetailan dari seluruh *chunk* yang dibuat.

Proses pemantauan perubahan kondisi pemain dijalankan setiap *frame* untuk mengetahui adakah perubahan terhadap *frame* sebelumnya. Hal ini dilakukan dengan cara membandingkan posisi terakhir pemain dengan posisi pemain saat ini. Jika pemain sudah melewati batas tertentu maka akan dilakukan perubahan kondisi baru. Proses ini ditunjukkan oleh fungsi *UpdateTrackingReference*.

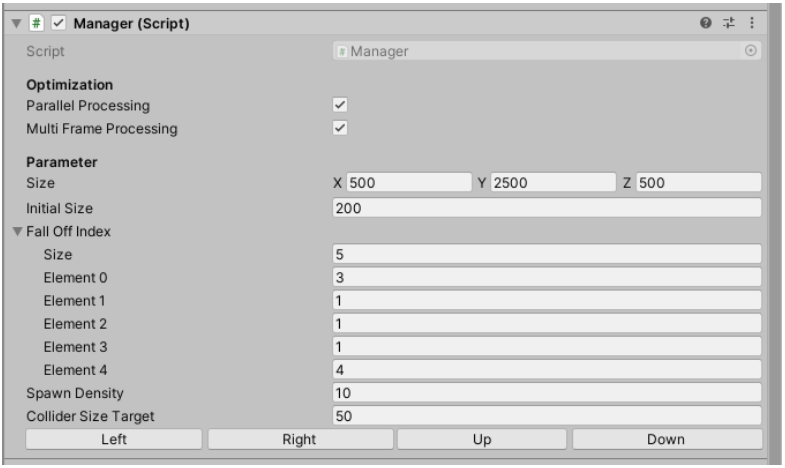
Jika terjadi perubahan kondisi ke kondisi yang baru maka akan dilakukan pengevaluasian kondisi saat ini. Proses ini berlangsung pada fungsi *OnPlayerPositionChanged*. Pada proses ini dilakukan komputasi terhadap *chunk* yang sudah tidak lagi dipakai untuk dihapuskan. Setelah itu dilakukan permintaan perubahan terhadap *chunk* yang sudah ada untuk mengganti ke resolusi yang diinginkan. Jika *chunk* belum pernah di proses maka akan dibuat sebuah *chunk* baru.

Setelah *chunk* baru berhasil dibuat maka *chunk* tersebut akan di diproses lebih lanjut untuk di integrasikan ke dalam dunia permainan. Proses pengintegrasian ini dilakukan di *thread* utama dengan cara membagi waktu pemrosesan. Proses pengintegrasian meliputi pembuatan *game object*, dan peletakan ke dalam dunia permainan. Proses ini dilakukan oleh fungsi *ProcessConstructorResponse*. Gambar 4.2 menunjukkan hasil akhir dari proses ini.



Gambar 4.2 Menunjukkan hasil akhir dari proses pembuatan.

Kelas *Manager* ini memiliki beberapa parameter yang dapat diganti untuk dapat mengubah hasil pembuatan secara umum. Gambar 4.3 menunjukkan seluruh parameter yang terdapat pada kelas *Manager*. Tabel 4.2 menjelaskan mengenai keterangan parameter tersebut.



Gambar 4.3 Parameter yang terdapat pada kelas *Manager*.

Tabel 4.2 Penjelasan parameter pada kelas *Manager*.

Nama	Deskripsi
Parallel Processing	Mengaktifkan atau menonaktifkan pemrosesan secara paralel
Multi Frame Processing.	Mengaktifkan atau menonaktifkan pemrosesan antar *frame*
Size	Ukuran *chunk* yang akan dibuat
Initial Size	Resolusi awal (maksimal) yang akan digunakan untuk setiap *chunk*
Fall Off Index	Jumlah *chunk* tiap tahap resolusi
Spawn Density	Jarak antar objek yang akan dievaluasi
Collider Size Target	Resolusi maksimal yang akan digunakan sebagai *collider*

```
1. public class Manager : MonoBehaviour {
2.
3.     [Header("Optimization")]
4.     [SerializeField] private bool parallelProcessing;

5.     [SerializeField] private bool multiFrameProcessin
   g;
6.
7.     [Header("Parameter")]
8.     [SerializeField] private Vector3 size;
9.     [SerializeField] private int initialSize;
10.        [SerializeField] private int[] fallOffIndex;
11.        [SerializeField] private float spawnDensity;
12.        [SerializeField] private int colliderSizeTarge
   t;
13.        private Transform trackingReference;
14.
15.        private Vector2Int playerPosition;
16.        private StateWatcher<Vector2Int> stateWatcher;

17.        private ITimedTaskRunner timedTaskRunner;
18.        private MonoGameObjectPool terrainMeshObjectPo
   ol;
19.        private Constructor constructor;
20.        private PropSpawner propSpawner;
21.        private SeaManipulator seaManipulator;
22.
23.        private Mutex constructorMutex;
24.        private Mutex terrainStateMutex;
25.        private Queue<ConstructorResponse> constructor
   Responses;
26.
27.        private Dictionary<Vector2Int, TerrainMeshStat
   e> terrainStates;
28.
29.        private int maxFalloff;
30.
31.        public Vector3 Size { get => size; }
32.
```

```
33.         public bool IsParallelProcessingEnabled { get
    => parallelProcessing; }
34.         public bool IsMultiFrameProcessingEnabled { ge
    t => multiFrameProcessing; }
35.
36.         public Manager() {
37.             stateWatcher = new StateWatcher<Vector2Int
    >(OnPlayerPositionChanged);
38.             constructorResponses = new Queue<Construct
    orResponse>();
39.             terrainStates = new Dictionary<Vector2Int,
     TerrainMeshState>();
40.             constructorMutex = new Mutex();
41.             terrainStateMutex = new Mutex();
42.         }
43.
44.         private void Awake() {
45.             TimedTaskRunner.IsTimedRun = multiFramePro
    cessing;
46.             ParallelRunner.IsMultiThreaded = parallelP
    rocessing;
47.
48.             timedTaskRunner = RenBehaviour.TraceCompon
    ent<MonoTimedTaskRunner>(transform, "Variable/TimedTa
    skRunner");
49.             terrainMeshObjectPool = RenBehaviour.Trace
    Component<MonoGameObjectPool>(transform, "Variable/Te
    rrainMeshPooler");
50.             constructor = RenBehaviour.TraceComponent<
    Constructor>(transform, "Variable/Constructor");
51.             stateWatcher.SetState(Vector2Int.zero);
52.             spawnDensity = Mathf.Clamp(spawnDensity, 1
    , initialSize);
53.             propSpawner = new PropSpawner(size, initia
    lSize, spawnDensity, timedTaskRunner);
54.
55.             maxFalloff = 0;
56.             int fallOffResolution = initialSize;
57.             int colliderReference = 99999;
58.             int finalColliderSize = -1;
59.             for (var i = 0; i < fallOffIndex.Length; i
    ++) {
60.                 maxFalloff += fallOffIndex[i];
```

```
61.                 fallOffIndex[i] = maxFalloff;
62.                 var colliderSizeRef = Mathf.Abs(fallOf
   fResolution - colliderSizeTarget);
63.                 if (colliderSizeRef < colliderReferenc
   e) {
64.                     colliderReference = colliderSizeRe
   f;
65.                     finalColliderSize = fallOffResolut
   ion;
66.                 }
67.                 fallOffResolution /= 2;
68.             }
69.             colliderSizeTarget = finalColliderSize;
70.
71.
72.         }
73.
74.         private void Start() {
75.             InitializeConstruction();
76.             var generator = RenBehaviour.TraceComponen
   t<Transform>(transform, "Generator");
77.             for (int i = 0; i < generator.childCount;
   i++) {
78.                 var component = generator.GetChild(i).
   GetComponent<IPropSpawnerEvaluator>();
79.                 if (component != null) {
80.                     propSpawner.AddEvaluator(component
   );
81.                 }
82.             }
83.             seaManipulator = RenBehaviour.TraceCompone
   nt<SeaManipulator>(transform, "Sea");
84.             seaManipulator.SetSeaParameter(Size.x * 1.
   2f, -(0.5f - constructor.SeaLevel) * Size.y);
85.         }
86.
87.         private void Update() {
88.             if (trackingReference != null) {
89.                 UpdateTrackingReference();
90.             }
91.             TimedTaskRunner.IsTimedRun = multiFramePro
   cessing;
```

```
92.             ParallelRunner.IsMultiThreaded = parallelP
   rocessing;
93.             ProcessConstructorResponse();
94.         }
95.
96.         private void UpdateTrackingReference() {
97.             var trackingPosition = new Vector2(trackin
   gReference.position.x, trackingReference.position.z);

98.             var xDiff = Mathf.Abs(trackingPosition.x /
    size.x - playerPosition.x);
99.             var yDiff = Mathf.Abs(trackingPosition.y /
    size.z - playerPosition.y);
100.            if (xDiff > 0.75f || yDiff > 0.75f) {
101.                playerPosition.x = Mathf.RoundToInt(tr
   ackingPosition.x / size.x);
102.                playerPosition.y = Mathf.RoundToInt(tr
   ackingPosition.y / size.z);
103.                stateWatcher.SetState(playerPosition);

104.                var seaManipulatorTransform = seaManip
   ulator.transform;
105.                seaManipulatorTransform.position = new
    Vector3(trackingReference.position.x, seaManipulator
   Transform.position.y, trackingReference.position.z);

106.            }
107.        }
108.
109.        public void SetTrackingReference(Transform tra
   cking) {
110.            trackingReference = tracking;
111.        }
112.
113.        private void OnPlayerPositionChanged(Vector2In
   t pos) {
114.            terrainStateMutex.WaitOne();
115.            var toBeDeleted = new LinkedList<TerrainMe
   shState>();
116.            foreach (var terrainState in terrainStates
   ) {
117.                var offset = terrainState.Key - pos;
118.                var absX = Mathf.Abs(offset.x);
```

```
119.                var absY = Mathf.Abs(offset.y);
120.                if (absX + absY >= maxFalloff) {
121.                    toBeDeleted.AddLast(terrainState.V
    alue);
122.                }
123.            }
124.            foreach (var node in toBeDeleted) {
125.                node.Dispose();
126.                terrainStates.Remove(node.Coordinate);

127.            }
128.
129.            var fallOffLimit = maxFalloff;
130.            for (var y = -
    fallOffLimit; y <= fallOffLimit; y++) {
131.                for (var x = -
    fallOffLimit; x <= fallOffLimit; x++) {
132.                    var absX = Math.Abs(x);
133.                    var absY = Math.Abs(y);
134.                    if (absX + absY < maxFalloff) {
135.                        var offset = pos + new Vector2
    Int(x, y);
136.                        var division = 1;
137.                        for (int i = 0; i < fallOffInd
    ex.Length; i++) {
138.                            if (absX + absY < fallOffI
    ndex[i]) {
139.                                break;
140.                            }
141.                            division *= 2;
142.                        }
143.                        TerrainMeshState terrainState;

144.                        if (terrainStates.ContainsKey(
    offset)) {
145.                            terrainState = terrainStat
    es[offset];
146.                        }
147.                        else {
148.                            terrainState = new Terrain
    MeshState(
149.                                offset,
```

```
150.                                    terrainMeshObjectPool,

151.                                    propSpawner,
152.                                    OnTerrainStateNeedRebu
   ilt);
153.                                terrainStates.Add(offset,
   terrainState);
154.                            }
155.                            var finalResolution = initialS
   ize / division;
156.                            terrainState.SetResolution(fin
   alResolution);
157.                        }
158.                    }
159.                }
160.                propSpawner.SetReferencePosition(pos, fall
   OffIndex[0]);
161.                terrainStateMutex.ReleaseMutex();
162.            }
163.
164.            private void InitializeConstruction() {
165.
166.                ParallelRunner.IsMultiThreaded = false;
167.                TimedTaskRunner.IsTimedRun = false;
168.
169.                var fallOffLimit = maxFalloff;
170.                for (var y = -
   fallOffLimit; y <= fallOffLimit; y++) {
171.                    for (var x = -
   fallOffLimit; x <= fallOffLimit; x++) {
172.                        var absX = Math.Abs(x);
173.                        var absY = Math.Abs(y);
174.                        if (absX + absY < maxFalloff) {
175.                            var offset = new Vector2Int(x,
    y);
176.                            TerrainMeshState terrainState;

177.                            if (terrainStates.ContainsKey(
   offset)) {
178.                                terrainState = terrainStat
   es[offset];
179.                            }
180.                            else {
```

```
181.                            terrainState = new Terrain
    MeshState(
182.                                offset,
183.                                terrainMeshObjectPool,

184.                                propSpawner,
185.                                OnTerrainStateNeedRebu
    ilt);
186.                            terrainStates.Add(offset,
    terrainState);
187.                        }
188.                        constructor.RequestRebuild(new
     ConstructorRequest(colliderSizeTarget, offset, size,
     OnInitConstructorResponse));
189.                    }
190.                }
191.            }
192.
193.            ParallelRunner.IsMultiThreaded = IsParalle
    lProcessingEnabled;
194.            TimedTaskRunner.IsTimedRun = IsMultiFrameP
    rocessingEnabled;
195.
196.            stateWatcher.ForceUpdate();
197.        }
198.
199.        private void OnTerrainStateNeedRebuilt(Vector2
    Int position, int resolution) {
200.            var offset = position - playerPosition;
201.            var absX = Math.Abs(offset.x);
202.            var absY = Math.Abs(offset.y);
203.            if (absX + absY < maxFalloff) {
204.                var division = 1;
205.                for (int i = 0; i < fallOffIndex.Lengt
    h; i++) {
206.                    if (absX + absY < fallOffIndex[i])
     {
207.                        break;
208.                    }
209.                    division *= 2;
210.                }
```

```
211.                constructor.RequestRebuild(new Constru
    ctorRequest(initialSize / division, position, size, O
    nConstructorResponse));
212.            }
213.        }
214.
215.
216.        private void OnInitConstructorResponse(Constru
    ctorResponse response) {
217.            constructorMutex.WaitOne();
218.            constructorResponses.Enqueue(response);
219.            constructorMutex.ReleaseMutex();
220.            ProcessConstructorResponse();
221.        }
222.
223.        private void OnConstructorResponse(Constructor
    Response response) {
224.            constructorMutex.WaitOne();
225.            constructorResponses.Enqueue(response);
226.            constructorMutex.ReleaseMutex();
227.        }
228.
229.        private void ProcessConstructorResponse() {
230.            terrainStateMutex.WaitOne();
231.            constructorMutex.WaitOne();
232.            while (constructorResponses.Count > 0) {
233.                var response = constructorResponses.De
    queue();
234.                if (terrainStates.ContainsKey(response
    .Request.Position)) {
235.                    var terrainMesh = terrainMeshObjec
    tPool.Acquire().GetComponent<TerrainMesh>();
236.                    terrainMesh.SetHeightMap(response.
    NoiseMap);
237.                    var reqPosition = response.Request
    .Position;
238.                    timedTaskRunner.AddTask(new ApplyT
    errainMesh(terrainMesh, response));
239.                    if (colliderSizeTarget == response
    .Request.Resolution || response.Request.Resolution <=
     colliderSizeTarget) {
240.                        timedTaskRunner.AddTask(new Ap
    plyTerrainCollider(terrainMesh, response.Mesh));
```

```
241.                    }
242.                    else {
243.                        if (!terrainStates.ContainsKey
   (reqPosition)) {
244.                            throw new Exception(string
   .Format("Requested to generate mesh on non existing b
   ase, {0}", reqPosition));
245.                        }
246.                        var terrainState = terrainStat
   es[reqPosition];
247.                        var colliderMesh = terrainStat
   e.GetTerrainMesh(colliderSizeTarget);
248.                        timedTaskRunner.AddTask(new Ap
   plyTerrainCollider(terrainMesh, colliderMesh.Mesh));

249.                    }
250.                    terrainStates[reqPosition].SetTerr
   ain(response.Request.Resolution, terrainMesh);
251.                }
252.            }
253.            constructorMutex.ReleaseMutex();
254.            terrainStateMutex.ReleaseMutex();
255.        }
256.
257.        public void MovePlayer(Vector2Int direction) {

258.            playerPosition += direction;
259.            stateWatcher.SetState(playerPosition);
260.        }
261.
262.        public void SetParallelProcessingFlag(bool val
   ue) {
263.            parallelProcessing = value;
264.        }
265.
266.        public void SetMultiFrameProcessingFlag(bool v
   alue) {
267.            multiFrameProcessing = value;
268.        }
269.    }
```

Kode Sumber 4.1 Implementasi dari kelas *Manager*.

4.3 Implementasi Constructor

Kelas ini bertugas untuk mengatur proses pembuatan sebuah *chunk*. Kelas ini juga bertugas untuk komputasi aturan warna yang digunakan untuk memberikan tekstur warna berdasarkan ketinggian tertentu yang nantinya akan digunakan oleh kelas lain. Proses penentuan aturan warna ini di komputasi pada fungsi *Awake*. Tidak hanya itu, kelas ini juga digunakan untuk menginisialisasi *pipeline* yang akan digunakan untuk membuat *noise map*. Proses penginisialisasi *pipeline* terjadi pada fungsi *StartNoise*.

Kelas ini juga berfungsi untuk menginisialisasi komponen pembuatan *terrain* lainnya yaitu *NoiseMapGenerator*, *MeshGenerator* dan juga *MeshMaterializer*. Hampir semua proses yang terjadi di kelas ini dikerjakan secara asinkronus di *thread* terpisah. Proses inisialisasi komponen yang dibutuhkan ini terjadi pada *constructor* kelas ini dan juga fungsi *Awake*.

Kelas ini memiliki beberapa parameter yang dapat diatur untuk mengubah hasil yang akan dihasilkan. Gambar 4.4 menunjukkan beberapa parameter yang dapat diatur pada kelas *Constructor* ini. Tabel 4.3 Menjelaskan tentang parameter yang berada pada kelas *Constructor* ini.

Gambar 4.4 Parameter yang terdapat pada kelas *Constructor*.

Tabel 4.3 Penjelasan parameter pada kelas *Constructor*.

Nama	Deskripsi
Method	Metode pengambilan sampel *noise*
Scale	Skala dari *noise* yang akan di sampel
H Value	Nilai korelasi proses pada *FBM*
Octave	Iterasi pengambilan sampel pada *FBM*
Default Material	Basis material yang akan digunakan
Base Materializer Size	Ukuran basis yang digunakan untuk membuat material
Base Materializer Resolution	Resolusi yang digunakan untuk membuat material

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using Rendoru.Scheduling;
5.  using NoiseGenerator;
6.  using NoiseGenerator.Node;
7.  using System;
8.  using Rendoru.Unity;
9.  using TerrainGenerator.Pipeline;
10.
11. namespace TerrainGenerator {
12.
13.     [Serializable]
14.     public class MaterializerColorData {
15.         public Color color;
16.         public float limit;
17.     }
18.
19.     public enum NoiseSamplerMethod {
20.         Simplex,
21.         Perlin
22.     }
23.
24.     /// <summary>
25.     /// Capable of performing construction of noise
        map and mesh
26.     /// </summary>
27.     public class Constructor : MonoBehaviour {
28.
29.         [SerializeField] private NoiseSamplerMethod
        method;
30.         [SerializeField] private float scale, hValue
        ;
31.         [SerializeField] private int octave;
32.         [SerializeField] private Material defaultMat
        erial;
33.
34.         [SerializeField] private int baseMaterialize
        rSize;
35.         [SerializeField] private int baseMaterialize
        rResolution;
```

```
36.         [SerializeField] private MaterializerColorDa
    ta[] colorData;
37.
38.
39.         private HorizontalPipeline pipeline;
40.
41.         private NoiseMapGenerator noiseMapGenerator;

42.         private MeshGenerator meshGenerator;
43.         private MeshMaterializer meshMaterializer;

44.         private ParallelRunner parallelRunner;
45.
46.         public float SeaLevel { get; private set; }

47.
48.         public Constructor() {
49.             parallelRunner = ParallelRunner.GetGloba
    lInstance();
50.             pipeline = new HorizontalPipeline();
51.             noiseMapGenerator = new NoiseMapGenerato
    r(pipeline);
52.         }
53.
54.         private void Awake() {
55.             var timedTaskRunner = RenBehaviour.Trace
    Component<MonoTimedTaskRunner>(transform, "../Tim
    edTaskRunner");
56.             meshGenerator = new MeshGenerator(timedT
    askRunner);
57.             SeaLevel = colorData[0].limit;
58.
59.             var materializerColor = new MeshMaterial
    izerColor[colorData.Length];
60.             for (int i = 0; i < materializerColor.Le
    ngth; i++) {
61.                 materializerColor[i] = new MeshMater
    ializerColor(colorData[i].color, colorData[i].lim
    it);
62.             }
63.             meshMaterializer = new MeshMaterializer(

64.                 baseMaterializerSize,
```

```
65.                 baseMaterializerResolution,
66.                 defaultMaterial,
67.                 parallelRunner,
68.                 timedTaskRunner,
69.                 materializerColor);
70.
71.             StartNoise();
72.         }
73.
74.         private void StartNoise() {
75.             if (method == NoiseSamplerMethod.Simplex
    ) {
76.                 scale *= 50;
77.             }
78.             var sampler = GetSampler(130698);
79.             var landScapeTerrain = new LandscapeTerr
    ain(scale, hValue, octave, sampler);
80.
81.             var verticalPipeline = new VerticalPipel
    ine();
82.
83.
84.             var mountainLandSampler = GetSampler(123
    45);
85.             var mountainLandTerrain = new LandscapeT
    errain(scale, hValue, octave, mountainLandSampler
    );
86.             var mountainTerrain = new MountainTerrai
    n(mountainLandTerrain);
87.
88.
89.             var seaLandSampler = GetSampler(54321);

90.             var seaLandTerrain = new LandscapeTerrai
    n(scale, hValue, octave, seaLandSampler);
91.             var seaTerrain = new SeaFloorTerrain(sea
    LandTerrain);
92.
93.             verticalPipeline.AddPipeline(mountainTer
    rain);
94.             verticalPipeline.AddPipeline(seaTerrain)
    ;
95.
```

```
96.
97.             pipeline.Add(landScapeTerrain);
98.             pipeline.Add(verticalPipeline);
99.         }
100.
101.        private INoiseSampler GetSampler(int seed) {

102.            INoiseSampler sampler;
103.            if (method == NoiseSamplerMethod.Simplex
        ) {
104.                sampler = new SimplexNoiseSampler(se
        ed);
105.            }
106.            else {
107.                sampler = new PerlinNoiseSampler(see
        d);
108.            }
109.            return sampler;
110.        }
111.
112.        public void RequestRebuild(ConstructorReques
        t request) {
113.            parallelRunner.AddTask(new _ConstructorP
        arallelRunner(new _ConstructorData(request), OnBu
        ildMap));
114.        }
115.
116.        private void OnBuildMap(_ConstructorParallel
        Runner constructRequest) {
117.            var request = constructRequest.Data.Requ
        est;
118.            var generateNoiseMapRequest = new _Const
        ructorRequestGenerateNoiseMap(constructRequest.Da
        ta, OnGenerateMesh);
119.            noiseMapGenerator.GenerateNoiseMap(new N
        oiseMapGeneratorRequest(request.Position, request
        .Resolution, generateNoiseMapRequest.OnGenerateNo
        iseMapDone));
120.        }
121.
122.        private void OnGenerateMesh(_ConstructorRequ
        estGenerateNoiseMap noiseMapRequest, NoiseMapGene
        ratorResponse response) {
```

```
123.            var data = noiseMapRequest.Data;
124.            data.NoiseMap = response.NoiseMap;
125.            var request = data.Request;
126.            var meshGeneratorRequest = new _Construc
      torRequestGenerateMesh(data, OnGenerateTexture);

127.            meshGenerator.GenerateMesh(new MeshGener
      atorRequest(data.NoiseMap.Map, request.Size, mesh
      GeneratorRequest.OnGenerateMeshDone));
128.        }
129.
130.        private void OnGenerateTexture(_ConstructorR
      equestGenerateMesh generateMeshRequest, MeshGener
      atorResponse response) {
131.            var data = generateMeshRequest.Data;
132.            data.Mesh = response.Mesh;
133.            var meshMaterializerRequest = new _Const
      ructorRequestMaterializeMesh(data, OnFinished);

134.            meshMaterializer.Materialize(new MeshMat
      erializerRequest(data.NoiseMap, meshMaterializerR
      equest.OnMaterializeMeshDone));
135.
136.        }
137.
138.        private void OnFinished(_ConstructorRequestM
      aterializeMesh materializeMeshRequest, MeshMateri
      alizerResponse response) {
139.            var data = materializeMeshRequest.Data;

140.            data.Material = response.Material;
141.            data.Texture = response.Texture;
142.            var request = data.Request;
143.            request.OnResponse(new ConstructorRespon
      se(request, data.NoiseMap, data.Mesh, data.Materi
      al, data.Texture));
144.        }
145.    }
146.
147.
148.    /*** SOME CLASS ARE HIDDEN ***/
149. }
```

Kode Sumber 4.2 Implementasi kelas *Constructor*.

4.4 Implementasi NoiseMapGenerator

Kelas ini bertugas untuk membuat *noise map* yang akan digunakan sebagai landasan pemrosesan ke depannya. Pemrosesan *noise map* dilakukan dengan cara menjalankan *pipeline* untuk membuat dan memodifikasi hasil dari tiap *noise*. Proses ini murni berjalan secara paralel untuk dapat mempercepat waktu pemrosesan dan tidak mengganggu *thread* utama. Proses pembuatan *noise map* ditunjukkan oleh fungsi *GenerateNoiseMapData*. Kelas ini juga memanfaatkan kelas *NoiseMapGenerator* pada namespace *NoiseGenerator* untuk membuat *raw noise* yang akan digunakan oleh kelas ini.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using System;
5.  using Rendoru.Scheduling;
6.  using Rendoru.Unity;
7.  using NoiseGenerator.Node;
8.  using Rendoru.Collection;
9.
10. namespace TerrainGenerator {
11.
12.   public class NoiseMap {
13.     public NoiseMap(Vector2Int position, float[][] map) {
14.       Position = position;
15.       Map = map;
16.     }
17.
18.     public Vector2Int Position { get; }
19.
20.     public float[][] Map { get; }
21.   }
22.   public class NoiseMapGenerator {
23.     private HorizontalPipeline pipeline;
```

```
24.         private ParallelRunner parallelRunner;
25.
26.         public NoiseMapGenerator(HorizontalPipeline pipeline) {
27.             this.pipeline = pipeline;
28.             parallelRunner = ParallelRunner.GetGlobalInstance();
29.         }
30.
31.         public void GenerateNoiseMap(NoiseMapGeneratorRequest
    request) {
32.             parallelRunner.AddTask(new _NoiseMapGeneratorParallel
    Runner(request, OnGenerateNoiseMap));
33.         }
34.
35.         private void OnGenerateNoiseMap(_NoiseMapGeneratorPar
    allelRunner runner) {
36.             var request = runner.Request;
37.             var noiseMap = GenerateNoiseMapData(new Vector3Int(re
    quest.Position.x, request.Position.y, request.Resolution));
38.             request.OnResponse(new NoiseMapGeneratorResponse(re
    quest, noiseMap));
39.         }
40.
41.         private NoiseMap GenerateNoiseMapData(Vector3Int data) {

42.             //data.Z contains resolution
43.             var position = new Vector2Int(data.x, data.y);
44.             var offset = new Vector2(1f, 1f) / data.z;
45.             var startPosition = position - offset * 0.51f; //Computing fr
    om N - 1 due to mesh need to be combined to eliminate seam
46.             var result = pipeline.Generate(
47.                 new Vector2Int(
48.                     data.z,
49.                     data.z),
50.                 startPosition,
51.                 Vector2.one + position + offset * 0.51f);
52.             return new NoiseMap(position, result);
53.         }
54.     }
55.
56.     /*** SOME CLASS ARE HIDDEN ***/
57. }
```

Kode Sumber 4.3 Implementasi kelas *NoiseMapGenerator* pada *namespace TerrainGenerator*.

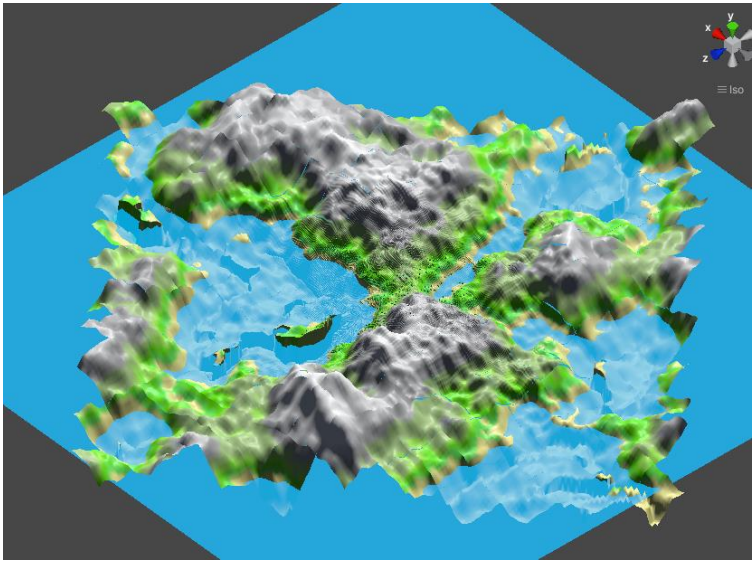
```
1.  using UnityEngine;
2.  using System.Collections;
3.  using Rendoru.Algorithm;
4.
5.
6.  namespace NoiseGenerator {
7.
8.    /// <summary>
9.    /// Capable of generating noise map
10.   /// </summary>
11.   public class NoiseMapGenerator : INoiseSampler {
12.
13.     private float scale;
14.     private INoiseSampler sampler;
15.
16.     public NoiseMapGenerator(float scale, INoiseSampler sampler
      ) {
17.       this.sampler = sampler;
18.       this.scale = scale;
19.     }
20.
21.     public float[][] Generate(Vector2Int size, Vector2 start, Vector
      2 end) {
22.       var result = ArrayUtil.Construct2DJaggedArray<float>(size.x,
      size.y);
23.       return Generate(result, start, end);
24.     }
25.
26.     public float[][] Generate(float[][] result, Vector2 start, Vector
      2 end) {
27.       var diff = (end - start);
28.       var offset = new Vector2();
29.       offset.x = diff.x / result.Length;
30.       for (int i = 0; i < result.Length; i++) {
31.         offset.y = diff.y / result[i].Length;
32.         for (int j = 0; j < result[i].Length; j++) {
33.           var posX = i * offset.x + start.x;
34.           var posY = j * offset.y + start.y;
```

```
35.                 result[i][j] = Sample(posX, posY);
36.             }
37.         }
38.         return result;
39.     }
40.
41.     public float Sample(Vector2 position) {
42.         return Sample(position.x, position.y);
43.     }
44.
45.     public float Sample(float xPosition, float yPosition) {
46.         var x = xPosition * scale;
47.         var y = yPosition * scale;
48.         return sampler.Sample(x, y);
49.     }
50.
51.   }
52. }
```

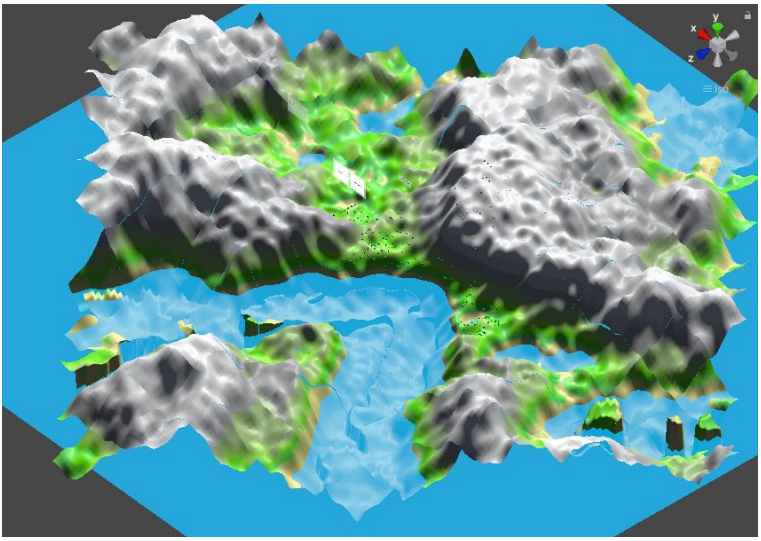
Kode Sumber 4.4 Implementasi kelas *NoiseMapGenerator* pada *namespace NoiseGenerator*.

4.5 Implementasi SimplexNoise

Kelas ini bertugas untuk melakukan sampling dari *noise* yang berbasis *simplex*. Dikarenakan *Unity* tidak memiliki implementasi noise berbasis *simplex* maka perlu ditambahkan pustaka pihak ketiga yang mengimplementasi *simplex noise*. Untuk itu dipilihlah implementasi oleh *Auburn* bernama *FastNoise* yang memiliki fungsi *simplex noise*. Dikarenakan kode sumber ini berbasis C++ maka perlu dilakukan pengadaptasian ke bahasa C#. Hal ini dilakukan dengan cara membuat *dynamic link library* untuk digunakan di *Unity*. Gambar 4.5 menunjukkan hasil sebelum menggunakan *simplex noise* dan gambar 4.6 menunjukkan hasil setelah menggunakannya.



Gambar 4.5 Hasil menggunakan *perlin noise.*



Gambar 4.6 Hasil menggunakan *simplex noise*.

```
1.  #include "pch.h"
2.  #include <string>
3.
4.  #define DLLExport __declspec(dllexport)
5.
6.  extern "C" {
7.
8.    DLLExport FastNoise* NewFastNoise(int seed) {
9.      return new FastNoise(seed);
10.   }
11.
12.   DLLExport void DeleteFastNoise(FastNoise* ptr) {
13.     delete ptr;
14.   }
15.
16.   DLLExport FN_DECIMAL GetSimplexNoiseFastNoise(FastNoise
    * ptr, FN_DECIMAL x, FN_DECIMAL y) {
17.     return ptr->GetSimplex(x, y);
18.   }
19.
20.   DLLExport float* GetLinearSimplexNoise(FastNoise* ptr, float*
     arrPtr, float offsetX, float offsetY, float step, int length) {
21.     for (auto i = 0; i < length; i++) {
22.       arrPtr[i + length] = ptr-
    >GetSimplex(offsetX + i * step, offsetY);
23.     }
24.     return arrPtr;
25.   }
26. }
```

Kode Sumber 4.5 Implementasi *interface simplex noise* pada bahasa C++.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System;
4.  using System.Runtime.InteropServices;
5.
6.
7.  namespace NoiseGenerator {
8.    public class SimplexNoiseSampler : INoiseSampler {
9.      private IntPtr mainPtr;
```

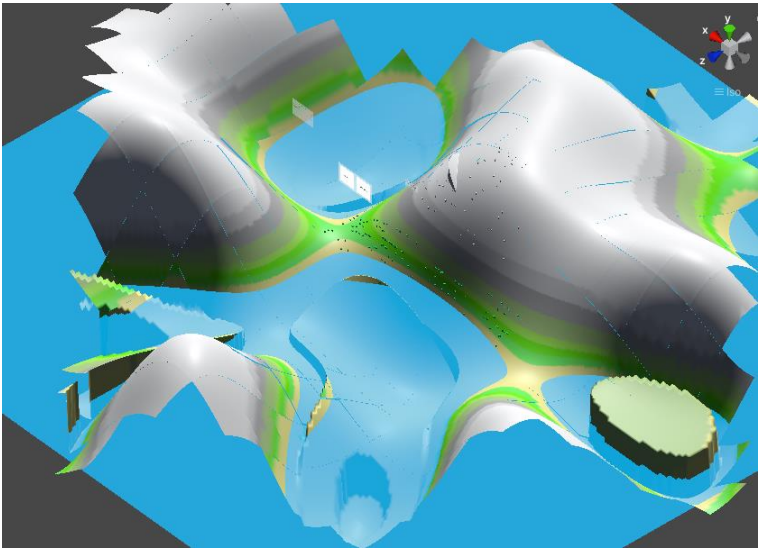
```
10.
11.     public SimplexNoiseSampler() : this(100) { }
12.
13.     public SimplexNoiseSampler(int seed) {
14.         mainPtr = NewFastNoise(seed);
15.     }
16.
17.     public float Sample(float x, float y) {
18.         var res = GetSimplexNoiseFastNoise(mainPtr, x, y);
19.         return (res - (-1)) / (1 - (-1));
20.     }
21.
22.     ~SimplexNoiseSampler() {
23.         DeleteFastNoise(mainPtr);
24.     }
25.
26.     [DllImport("FastNoiseUnity_x64")]
27.     private static extern IntPtr NewFastNoise(int seed);
28.
29.     [DllImport("FastNoiseUnity_x64")]
30.     private static extern void DeleteFastNoise(IntPtr ptr);
31.
32.     [DllImport("FastNoiseUnity_x64")]
33.     private static extern float GetSimplexNoiseFastNoise(IntPtr p
    tr, float x, float y);
34.
35.   }
36. }
```

Kode Sumber 4.6 Implementasi *interface simplex noise* pada bahasa C#.

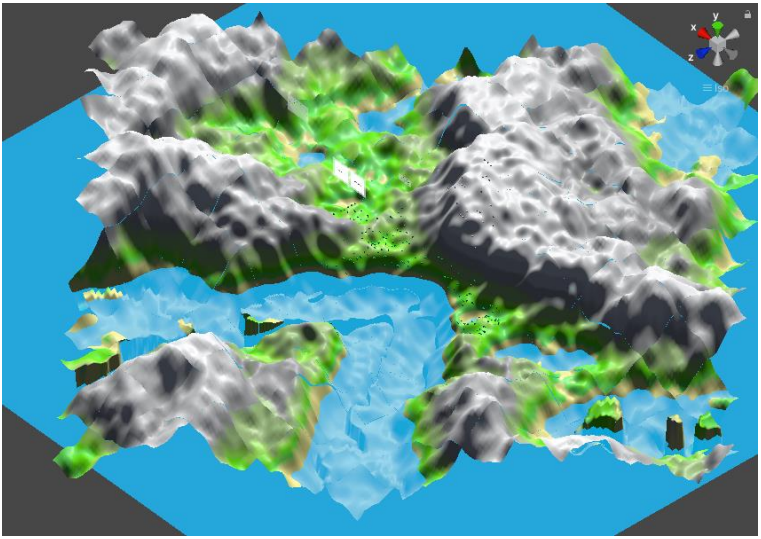
4.6 Implementasi FractionalBrownianMotion

Kelas ini bertugas untuk melakukan pengambilan sampel dengan menerapkan *Fractional Brownian Motion* untuk memberikan detail – detail kecil pada sampel tersebut. Pada *constructor* kelas ini dilakukan pengambilan sampel nilai minimum dan maksimum yang dapat diperoleh jika menggunakan metode ini. Hal ini diperlukan karena proses selanjutnya membutuhkan nilai di rentang 0 hingga 1. Proses komputasi sampel

dilakukan pada fungsi *Compute*. Gambar 4.7 menunjukkan hasil sebelum menerapkan *fractional brownian motion* dan gambar 4.8 menunjukkan hasil setelah menerapkannya.



Gambar 4.7 Hasil sebelum menggunakan *fractional brownian motion*.



Gambar 4.8 Hasil setelah menggunakan *fractional brownian motion*.

```
using UnityEngine;
using System.Collections;

namespace NoiseGenerator {
  public class FractionalBrownianMotion {

    private float exponent;
    private float maxValue;

    private INoiseSampler sampler;
    private int octave;

    /// <summary>
    ///
    /// </summary>
    /// <param name="hValue">How much noise constributed to current sample, higher value means lower difference</param>
    /// <param name="octave">How many iteration is performed, higher value yields detailed noise, High performance impact</param>
```

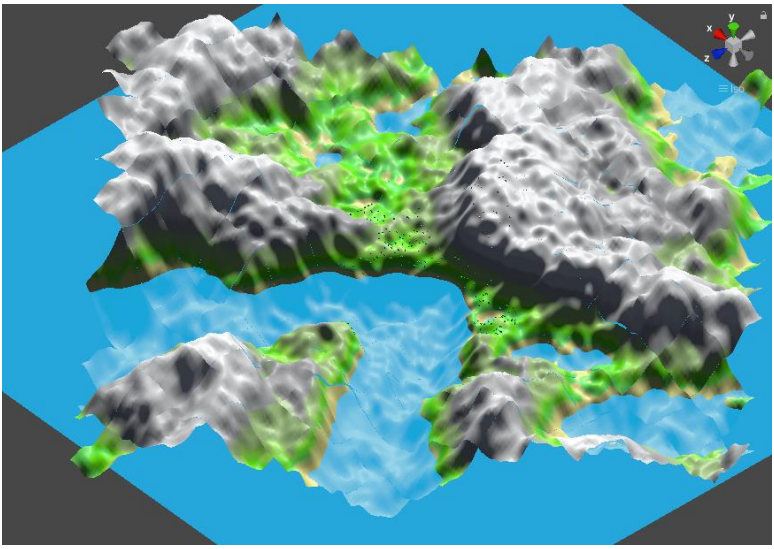
```
18.         /// <param name="sampler">Sampler to be used</param>

19.         public FractionalBrownianMotion(float hValue, int octave,
    INoiseSampler sampler) {
20.             this.sampler = sampler;
21.             this.octave = octave;
22.             exponent = Mathf.Pow(2, -hValue);
23.             float amplitude = 0.5f;
24.             maxValue = 0;
25.             for (int i = 0; i < octave; i++) {
26.                 maxValue += amplitude;
27.                 amplitude *= exponent;
28.             }
29.         }
30.
31.         public float Compute(float x, float y) {
32.             float result = 0f;
33.             float frequency = 1f;
34.             float amplitude = 0.5f;
35.             for (int i = 0; i < octave; i++) {
36.                 result += amplitude * sampler.Sample(frequency * x, fre
    quency * y);
37.                 frequency *= 2;
38.                 amplitude *= exponent;
39.             }
40.             return Mathf.InverseLerp(0f, maxValue, result); //To kee
    p values between 0 and 1
41.         }
42.     }
43. }
```

Kode Sumber 4.7 Implementasi kelas *FractionalBrownianMotion*.

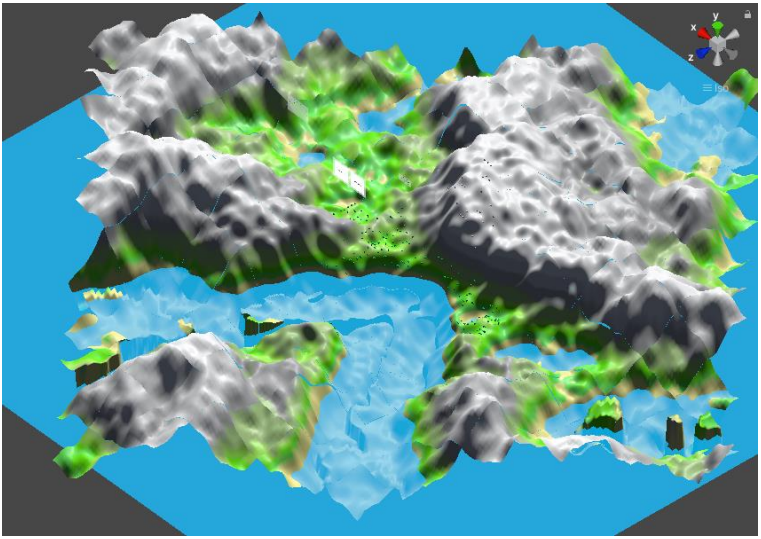
4.7 Implementasi Noise Pipeline

Terdapat dua jenis *pipeline* yang diimplementasikan yaitu *horizontal* dan *vertical pipeline*. Pada *horizontal pipeline* fungsi *Generate* hanya melakukan generasi terhadap seluruh data yang ada. Pada *vertical pipeline* fungsi *Generate* mengambil data secara satu – persatu lalu melanjutkannya ke *pipeline* berikutnya. Gambar

4.9 menunjukkan hasil sebelum ditambahkan *pipeline* dan gambar 4.10 menunjukkan hasil setelah ditambahkan *pipeline*.



Gambar 4.9 Hasil sebelum ditambahkan *pipeline*.



Gambar 4.10 Hasil setelah ditambahkan *pipeline*.

```
1.  using UnityEngine;
2.  using System.Collections.Generic;
3.  using System.Collections;
4.  using Rendoru.Algorithm;
5.
6.  namespace NoiseGenerator.Node {
7.      /// <summary>
8.      /// Capable of supporting generation of horizontal terrain
9.      /// Horizontal pipeline is processed by map by map resulting in
    knowledge with area in the map
10.     /// However when there are many map to be processed, then it c
    an be slow due to iteration time required
11.     /// So if possible please <see cref="IVerticalPipeline"/> or use de
    fault pipeline <see cref="VerticalPipeline"/>
12.     /// </summary>
13.     public class HorizontalPipeline : IHorizontalPipeline {
14.
15.         private LinkedList<IHorizontalPipeline> nodes;
16.
17.         public HorizontalPipeline() {
```

```
18.         nodes = new LinkedList<IHorizontalPipeline>();
19.     }
20.     public float[][] Generate(Vector2Int size, Vector2 start, Vector
    2 end) {
21.         var result = ArrayUtil.Construct2DJaggedArray<float>(size.x,
     size.y);
22.         return Generate(result, start, end);
23.     }
24.
25.     public float[][] Generate(float[][] map, Vector2 start, Vector2
    end) {
26.         foreach (var node in nodes) {
27.             map = node.Generate(map, start, end);
28.         }
29.         return map;
30.     }
31.
32.     public LinkedListNode<IHorizontalPipeline> Add(IHorizontal
    Pipeline node) {
33.         return nodes.AddLast(node);
34.     }
35.   }
36. }
```

Kode Sumber 4.8 Implementasi *horizontal pipeline*.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.
5.  namespace NoiseGenerator.Node {
6.
7.
8.    /// <summary>
9.    /// Support generation of Vertical Terrain Pipeline
10.   /// Vertical terrain are processed vertically, that means each coo
    rdinate is evaluated one by one in each pipeline
11.   /// Hopefully it will speedup execution since it doesn't have to co
    mpute loop for each pipeline
12.   /// </summary>
13.   public class VerticalPipeline : IHorizontalPipeline {
```

```
14.
15.     private LinkedList<IVerticalPipeline> pipelines;
16.
17.     public VerticalPipeline() {
18.         pipelines = new LinkedList<IVerticalPipeline>();
19.     }
20.
21.     public float[][] Generate(float[][] map, Vector2 start, Vector2
    end) {
22.         var position = new Vector2();
23.         var diff = end - start;
24.         diff.x = diff.x / map.Length;
25.         diff.y = diff.y / map.Length;
26.         for (int i = 0; i < map[0].Length; i++) {
27.             for (int j = 0; j < map.Length; j++) {
28.                 position.x = diff.x * j + start.x;
29.                 position.y = diff.y * i + start.y;
30.                 foreach (var pipeline in pipelines) {
31.                     map[j][i] = pipeline.Generate(map[j][i], position);
32.                 }
33.             }
34.         }
35.         return map;
36.     }
37.
38.     public LinkedListNode<IVerticalPipeline> AddPipeline(IVertic
    alPipeline pipeline) {
39.         return pipelines.AddLast(pipeline);
40.     }
41.   }
42. }
```

Kode Sumber 4.9 Implementasi *vertical pipeline*.

4.8 Implementasi MeshGenerator

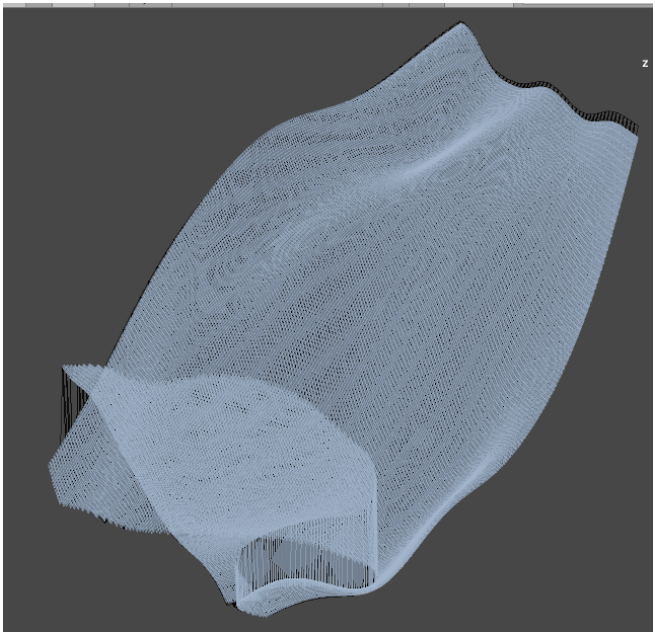
Kelas ini bertugas untuk membuat *mesh* atau rupa yang akan digunakan untuk merepresentasikan sebuah *noise map*. Pada proses ini terjadi beberapa proses yaitu komputasi *vertex*, komputasi *triangle*, komputasi *UV map* dan komputasi *normal map*. Untuk komputasi *normal map* dapat dilakukan secara otomatis oleh *Unity*

sehingga kita hanya perlu untuk komputasi *vertex*, *triangle* dan juga *UV map*.

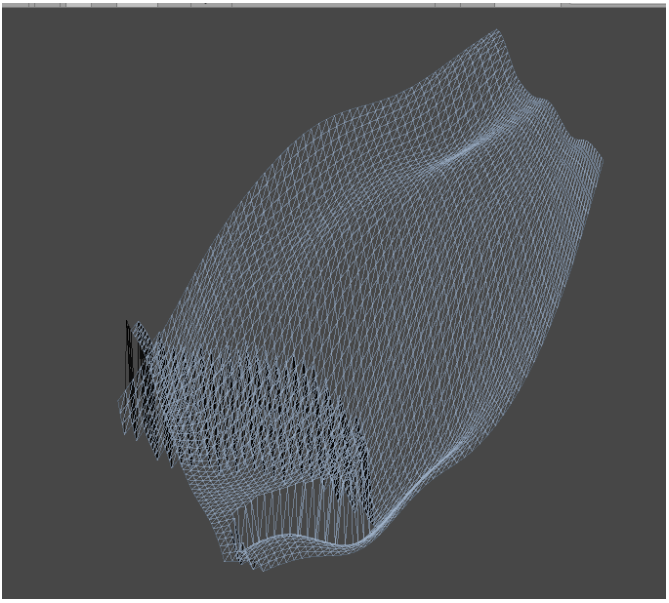
Proses komputasi *vertex*, *triangle* dan *UV map* dapat dilakukan secara paralel. Untuk komputasi *Vertex* dan *UV map* dapat dilihat pada fungsi *GenerateVerticesAndUVs*. Fungsi ini akan membuat kumpulan *vertex* dan titik *UV* secara bersamaan. Komputasi *UV map* diperlukan untuk digunakan untuk meletakkan tekstur pada *mesh* yang akan dibuat.

Proses berikutnya adalah pembuatan *triangle* dari kumpulan *vertex* yang telah dibuat. Kumpulan *vertex* tersebut akan di atur ulang untuk membuat sebuah *quad* dari 2 buah *triangle*. Proses pembuatan *triangle* ditunjukkan oleh fungsi *GenerateTriangles* dan proses pemetaan *quad* dilakukan oleh fungsi *MapQuad*. Proses ini juga dilaksanakan secara paralel.

Ketika semua data sudah tersedia maka proses selanjutnya adalah untuk membuat sebuah *mesh* berdasarkan data tersebut. Proses ini ditunjukkan oleh fungsi *ApplyVertAndTris* dan dieksekusi di *thread* utama. Proses ini juga melakukan komputasi *normal map* secara otomatis berdasarkan data yang ada. Gambar 4.11 menunjukkan *mesh* hasil pembuatan dengan resolusi 200 sedangkan gambar 4.12 menunjukkan *mesh* dengan resolusi 50.



Gambar 4.11 *Mesh* dengan resolusi sebesar 200.



Gambar 4.12 *Mesh* dengan resolusi sebesar 50.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using Rendoru.Scheduling;
5.  using System;
6.  using System.Threading;
7.  using Rendoru.Unity;
8.  using UnityEditor;
9.
10. namespace TerrainGenerator {
11.     /// <summary>
12.     /// Capable of generating terrain mesh from 2D f
    loat data
13.     /// </summary>
14.     public class MeshGenerator {
15.
16.         private ITimedTaskRunner timedTaskRunner;
17.         private ParallelRunner parallelRunner;
```

```
18.
19.         public MeshGenerator(ITimedTaskRunner timedT
    askRunner) {
20.             this.timedTaskRunner = timedTaskRunner;

21.             parallelRunner = ParallelRunner.GetGloba
    lInstance();
22.         }
23.
24.         public void GenerateMesh(MeshGeneratorReques
    t request) {
25.             parallelRunner.AddTask(new _MeshGenerato
    rParallelRunnerRequest(request, GenerateVertAndTris)
    );
26.         }
27.
28.         private void GenerateVertAndTris(MeshGenerat
    orRequest request) {
29.             var vertAndUv = GenerateVerticesAndUVs(r
    equest.Map, request.Size);
30.             var triangles = GenerateTriangels(reques
    t.Map);
31.             timedTaskRunner.AddTask(new _MeshGenerat
    orCombinerRequest(request, ApplyVertAndTris, vertAnd
    Uv.Item1, vertAndUv.Item2, triangles));
32.         }
33.
34.         private void ApplyVertAndTris(_MeshGenerator
    CombinerRequest request) {
35.             var mesh = new Mesh();
36.             var size = request.Request.Size;
37.             mesh.SetVertices(request.Vertices);
38.             mesh.SetTriangles(request.Tris, 0);
39.             mesh.SetUVs(0, request.UVs);
40.             mesh.RecalculateNormals();
41.             request.Request.OnResponse(new MeshGener
    atorResponse(request.Request, mesh));
42.         }
43.
44.         private ValueTuple<Vector3[], Vector2[]> Gen
    erateVerticesAndUVs(float[][] map, Vector3 size) {
```

```
45.             var vertices = new Vector3[map.Length *
    map[0].Length]; //Need clarification whether this ar
    ray is copied or not,
46.             var uv = new Vector2[map.Length * map[0]
    .Length]; //Need clarification whether this array is
     copied or not,
47.             for (int i = 0; i < map.Length; i++) {
48.                 var xPos = Mathf.InverseLerp(0, map.
    Length - 1, i);
49.                 for (int j = 0; j < map[i].Length; j
    ++) {
50.                     var yPos = Mathf.InverseLerp(0,
    map[i].Length - 1, j);
51.                     vertices[i + j * map[i].Length]
    = new Vector3(xPos * size.x, map[i][j] * size.y, yPo
    s * size.z);
52.                     uv[i + j * map[i].Length] = new
    Vector2(i / (float)map.Length, j / (float)map[i].Len
    gth);
53.                 }
54.             }
55.             return new ValueTuple<Vector3[], Vector2
    []>(vertices, uv);
56.         }
57.
58.         private int[] GenerateTriangels(float[][] ma
    p) {
59.             var sizeX = map.Length - 1;
60.             var sizeY = map[0].Length - 1;
61.             var triangles = new int[6 * map.Length *
     map[0].Length]; //Need clarification whether this a
    rray is copied or not
62.             var indexer = 0;
63.             for (int i = 0; i < sizeX; i++) {
64.                 var yLength = map[i].Length;
65.                 for (int j = 0; j < sizeY; j++) {
66.                     MapQuad(triangles, indexer, i, j
    , yLength);
67.                     indexer++;
68.                 }
69.             }
70.             return triangles;
71.         }
```

```
72.
73.         private void MapQuad(int[] triangles, int id
    x, int x, int y, int ySize) {
74.             var indexer = idx * 6;
75.             triangles[indexer++] = x + y * ySize;
76.             triangles[indexer++] = x + (y + 1) * ySi
    ze;
77.             triangles[indexer++] = x + 1 + y * ySize
    ;
78.
79.             triangles[indexer++] = x + 1 + y * ySize
    ;
80.             triangles[indexer++] = x + (y + 1) * ySi
    ze;
81.             triangles[indexer++] = x + 1 + (y + 1) *
     ySize;
82.         }
83.     }
84.
85.     /*** SOME CLASS ARE HIDDEN ***/
86. }
```

Kode Sumber 4.10 Implementasi kelas *MeshGenerator*.

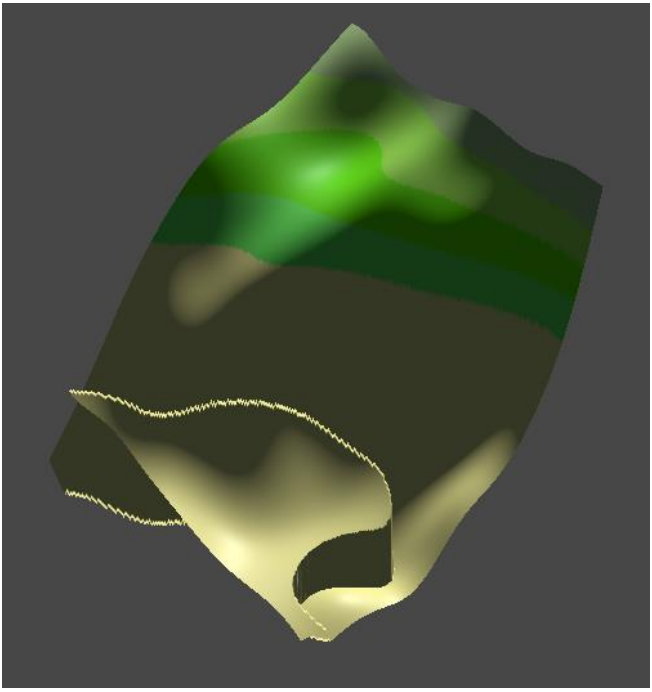
4.9 Implementasi MeshMaterializer

Kelas ini bertugas untuk membuat material yang akan digunakan pada *mesh* yang telah dibuat. Terjadi beberapa sub proses pada proses ini yaitu membuat warna dan juga membuat tekstur warna.

Proses pembuatan warna dilakukan oleh fungsi *GenerateColors*. Proses ini akan menentukan warna yang akan digunakan berdasarkan nilai dari *noise map* saat ini. Proses penentuan aturan warna yang digunakan ditunjukkan pada fungsi *GetColor*. Proses ini dieksekusi secara paralel.

Proses selanjutnya adalah proses pembuatan material. Pembuatan tekstur akan dibuat berdasarkan warna yang telah dibuat pada proses sebelumnya. Dikarenakan proses ini memanfaatkan fungsi yang dimiliki oleh *Unity* maka proses ini

dilaksanakan di *thread* utama. Proses ini ditunjukkan oleh fungsi *GenerateTexture*. Gambar 4.13 menunjukkan penerapan dari material yang terbuat.



Gambar 4.13 Penerapan material yang terbuat pada *mesh*.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System;
4.  using Rendoru.Scheduling;
5.  using Rendoru.Collection;
6.  using Rendoru.Algorithm;
7.
8.  namespace TerrainGenerator {
9.
10.     public class MeshMaterializerColor {
```

```
11.         public MeshMaterializerColor(Color color, fl
    oat limit) {
12.             this.Color = color;
13.             this.Limit = limit;
14.         }
15.
16.         public Color Color { get; }
17.
18.         public float Limit { get; }
19.     }
20.     public class MeshMaterializer {
21.
22.         private readonly int baseSize;
23.         private readonly int baseResolution;
24.         private readonly ITaskRunner taskRunner;
25.         private readonly ITimedTaskRunner timedTaskR
    unner;
26.         private readonly MeshMaterializerColor[] rul
    es;
27.         private readonly Material defaultMaterial;
28.
29.         public MeshMaterializer(
30.             int baseSize,
31.             int baseResolution,
32.             Material defaultMaterial,
33.             ITaskRunner taskRunner,
34.             ITimedTaskRunner timedTaskRunner,
35.             MeshMaterializerColor[] rules) {
36.
37.             this.baseResolution = baseResolution;
38.             this.baseSize = baseSize;
39.             this.taskRunner = taskRunner;
40.             this.timedTaskRunner = timedTaskRunner;

41.             this.rules = rules;
42.             this.defaultMaterial = defaultMaterial;

43.         }
44.
45.         public void Materialize(MeshMaterializerRequ
    est request) {
46.             taskRunner.AddTask(new _MeshMaterializer
    ParallelRunner(request, GenerateColors));
```

```
47.         }
48.
49.         private void GenerateColors(MeshMaterializer
    Request request) {
50.             var noiseMap = request.NoiseMap;
51.             var resolution = (float)noiseMap.Map.Len
    gth;
52.             var ratio = resolution / baseResolution;

53.             var actualSize = (int)(ratio * baseSize)
    ;
54.             var mappedArray = new Mapped2DArray<Colo
    r>(actualSize, actualSize);
55.             var noises = noiseMap.Map;
56.             for (var i = 0; i < actualSize; i++) {
57.                 for (var j = 0; j < actualSize; j++)
     {
58.                     var sampleI = (int)(i * ratio);

59.                     var sampleJ = (int)(j * ratio);

60.                     var height = noises[i][j];
61.                     mappedArray[i, j] = GetColor(hei
    ght);
62.                 }
63.             }
64.             timedTaskRunner.AddTask(new _MeshMateria
    lizerSetPixel(request, mappedArray, GenerateTexture)
    );
65.         }
66.
67.         private Color GetColor(float height) {
68.             var color = rules[0].Color;
69.             var offset = rules[0].Limit;
70.             for (int i = 1; i < rules.Length; i++) {

71.                 var rule = rules[i];
72.                 offset += rule.Limit;
73.                 if (offset > height) {
74.                     break;
75.                 }
76.                 color = rule.Color;
77.             }
```

```
78.             return color;
79.         }
80.
81.         private void GenerateTexture(MeshMaterialize
    rRequest request, Mapped2DArray<Color> colors) {
82.             var texture = new Texture2D(colors.First
    Size, colors.SecondSize);
83.             texture.wrapMode = TextureWrapMode.Clamp
    ;
84.             texture.filterMode = FilterMode.Trilinea
    r;
85.             texture.SetPixels(colors.Array);
86.             texture.Apply();
87.             var material = new Material(defaultMater
    ial);
88.             material.SetTexture("_BaseMap", texture)
    ;
89.             request.OnFinished(new MeshMaterializerR
    esponse(request, material, texture));
90.         }
91.     }
92.
93.     /*** SOME CLASS ARE HIDDEN ***/
94. }
```

Kode Sumber 4.11 Implementasi kelas *MeshMaterializer*.

4.10 Implementasi TerrainMeshState

Kelas ini bertugas untuk melakukan pemantauan pada *chunk* yang telah dibuat. Jika terjadi perubahan status pada *chunk* tersebut maka kelas ini bertugas untuk mengimplementasikan perubahan tersebut.

Untuk memberitahukan perubahan yang harus dilakukan pada kelas ini adalah dengan cara memanggil fungsi *SetResolution*. Fungsi ini akan memperbaharui status dari kelas ini dan jika terjadi perubahan status ke status yang baru maka kelas ini akan menerapkan perubahan tersebut. Jika perubahan yang dituju belum pernah diminta sebelumnya maka kelas ini akan melakukan permintaan pemrosesan terhadap *chunk* yang diminta dan menyimpannya. Namun jika sudah pernah diproses maka kelas ini

akan menerapkan perubahan tersebut berdasarkan permintaan sebelumnya.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using Rendoru.Algorithm;
4.  using Rendoru.Unity;
5.  using System;
6.  using System.Collections.Generic;
7.  using Generative;
8.  using Rendoru.Scheduling;
9.
10. namespace TerrainGenerator {
11.     public class TerrainMeshState : IDisposable {
12.
13.         private MonoGameObjectPool gameObjectPool;
14.         private Action<Vector2Int, int> onNeedRebuil
    d;
15.
16.         private Dictionary<int, TerrainMesh> terrain
    s;
17.         private StateWatcher<int> resolutionWatcher;

18.         private TerrainMesh activeTerrain;
19.         private PropSpawner propSpawner;
20.
21.
22.         public Vector2Int Coordinate { get; }
23.
24.
25.         public TerrainMeshState(
26.             Vector2Int coordinate,
27.             MonoGameObjectPool gameObjectPool,
28.             PropSpawner propSpawner,
29.             Action<Vector2Int, int> onNeedRebuild
30.             ) {
31.
32.             this.propSpawner = propSpawner;
33.             this.gameObjectPool = gameObjectPool;
34.             Coordinate = coordinate;
35.             this.onNeedRebuild = onNeedRebuild;
```

```
36.             resolutionWatcher = new StateWatcher<int
    >(OnResolutionChange);
37.             terrains = new Dictionary<int, TerrainMe
    sh>();
38.         }
39.
40.         public void Dispose() {
41.             if (activeTerrain != null) {
42.                 activeTerrain.gameObject.SetActive(f
    alse);
43.                 activeTerrain = null;
44.             }
45.             foreach (var terrain in terrains) {
46.                 gameObjectPool.Release(terrain.Value
    .gameObject);
47.             }
48.         }
49.
50.         public void SetTerrain(int resolution, Terra
    inMesh terrain) {
51.             if (activeTerrain != null) {
52.                 activeTerrain.gameObject.SetActive(f
    alse);
53.             }
54.             terrains[resolution] = terrain;
55.             activeTerrain = terrain;
56.             activeTerrain.gameObject.SetActive(true)
    ;
57.             propSpawner.Spawn(Coordinate, resolution
    , terrain.HeightMap);
58.         }
59.
60.         private void OnResolutionChange(int res) {
61.             if (terrains.ContainsKey(res)) {
62.                 if (activeTerrain != null) {
63.                     activeTerrain.gameObject.SetActi
    ve(false);
64.                 }
65.                 activeTerrain = terrains[res];
66.                 activeTerrain.gameObject.SetActive(t
    rue);
67.                 propSpawner.Spawn(Coordinate, res, a
    ctiveTerrain.HeightMap);
```

```
68.             }
69.             else {
70.                 onNeedRebuild(Coordinate, res);
71.             }
72.         }
73.
74.         public void SetResolution(int resolution) {
75.             resolutionWatcher.SetState(resolution);
76.         }
77.
78.         public TerrainMesh GetTerrainMesh(int resolution) {
79.             if (!terrains.ContainsKey(resolution)) {
80.                 throw new Exception(string.Format("Resolution couldn't be found on {0}", resolution));
81.             }
82.             return terrains[resolution];
83.         }
84.     }
85. }
```

Kode Sumber 4.12 Implementasi kelas *TerrainMeshState*.

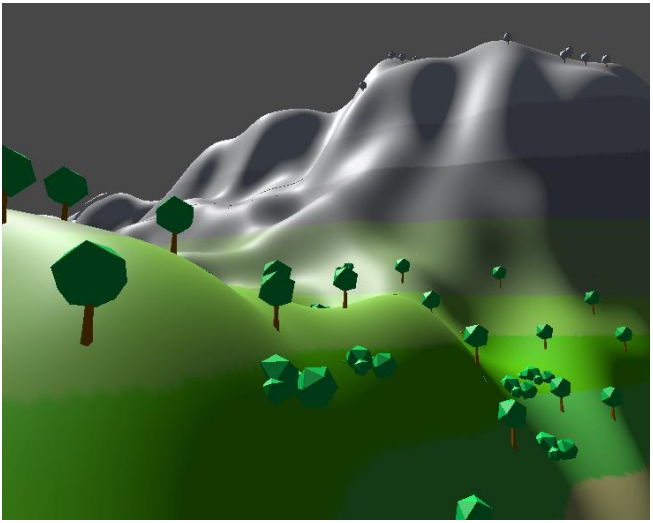
4.11 Implementasi PropSpawner

Kelas ini bertugas untuk melakukan peletakan objek yang akan diletakkan ke dalam dunia permainan. Objek – objek yang akan diletakkan ke dalam dunia permainan akan diletakkan jika memenuhi prasyarat dari *evaluator*. Kelas ini berhubungan dengan kelas *SpawnPropTask* yang digunakan untuk mengeksekusi proses evaluasi.

Kelas ini menentukan apakah perlu adanya evaluasi dititik ini dan *evaluator* mana yang perlu dipanggil di titik ini. Setiap *evaluator* akan diberikan kesempatan untuk mengevaluasi nilai yang diberikan oleh *evaluator* dan juga menentukan sendiri objek mana yang akan ditampilkan. Kelas ini juga bertugas untuk melakukan pemantauan pada seluruh objek yang ada dalam dunia

permainan untuk diambil dan digunakan kembali jika sudah tidak lagi digunakan.

Proses penentuan evaluasi di tentukan oleh fungsi *Spawn* di mana fungsi ini akan menyiapkan data untuk dilakukannya proses evaluasi. Proses evaluasi selanjutnya akan dialihkan ke kelas *SpawnPropTask* untuk dilakukan evaluasi. Proses evaluasi pada kelas tersebut akan dilakukan pada fungsi *Execute* di mana proses ini akan dilakukan di *thread* utama. Jika titik tersebut dievaluasi maka proses evaluasi selanjutnya akan dieksekusi oleh fungsi *Evaluate* pada kelas *PropSpawner*. Gambar 4.14 menunjukkan contoh penempatan pohon yang terbuat.



Gambar 4.14 Penempatan objek pohon.

```
1.  using UnityEngine;
2.  using System.Collections.Generic;
3.  using System.Collections;
4.  using System;
5.  using Generative;
6.  using Rendoru.Collection;
```

```
7.  using Rendoru.Unity;
8.  using Rendoru.Scheduling;
9.
10. namespace TerrainGenerator {
11.     public class PropSpawner {
12.         private Vector2Int referencePosition;
13.         private int referenceRadius;
14.
15.         private Vector3 distance;
16.         private int resolution;
17.         private ITimedTaskRunner timedTaskRunner;
18.         private Dictionary<Vector2Int, LinkedList<ID
    isposable>> props;
19.         private LinkedList<IPropSpawnerEvaluator> ev
    aluators;
20.         private float density;
21.
22.         private static int pressure;
23.
24.         public PropSpawner(Vector3 distance, int res
    olution, float density, ITimedTaskRunner timedTaskRu
    nner) {
25.             this.distance = distance;
26.             this.resolution = resolution;
27.             this.timedTaskRunner = timedTaskRunner;

28.             this.density = density;
29.             props = new Dictionary<Vector2Int, Linke
    dList<IDisposable>>();
30.             evaluators = new LinkedList<IPropSpawner
    Evaluator>();
31.         }
32.
33.         public void Spawn(Vector2Int position, int r
    esolution, NoiseMap noiseMap) {
34.             if (this.resolution != resolution) {
35.                 return;
36.             }
37.             var spawned = new LinkedList<IDisposable
    >();
38.             if (props.ContainsKey(position)) {
39.                 Clean(position);
40.             }
```

```
41.             props[position] = spawned;
42.             timedTaskRunner.AddTask(new SpawnPropTas
    k(spawned, noiseMap, distance, density, Evaluate, On
    SpawnFinished));
43.         }
44.
45.         private MonoDisposableGameObject Evaluate(fl
    oat height, int posX, int posY) {
46.             var iter = evaluators.First;
47.             var weirdConstant = 1.234177f; //Arbitra
    ryly picked
48.             while (iter != null) {
49.                 if ((int)(height * 100 * weirdConsta
    nt) % 2 > 0) {
50.                     var res = iter.Value.Evaluate(he
    ight, posX, posY);
51.                     if (res != null) {
52.                         return res;
53.                     }
54.                 }
55.                 weirdConstant *= weirdConstant;
56.                 iter = iter.Next;
57.             }
58.             return null;
59.         }
60.
61.         public void AddEvaluator(IPropSpawnerEvaluat
    or evaluator) {
62.             evaluators.AddLast(evaluator);
63.         }
64.
65.         private void Clean(Vector2Int position) {
66.             if (props.ContainsKey(position)) {
67.                 var toBeCleaned = props[position];
68.                 props.Remove(position);
69.                 timedTaskRunner.AddTask(new CleanPro
    pTask(toBeCleaned, position));
70.             }
71.         }
72.
73.         private void Clean(Vector2Int position, int
    radius) {
```

```
74.             var toBeCleaned = new LinkedList<Vector2
    Int>();
75.             foreach (var prop in props.Keys) {
76.                 var propPosition = prop;
77.                 if (Mathf.Abs(position.x - propPosit
    ion.x) + Mathf.Abs(position.y - propPosition.y) >= r
    adius) {
78.                     toBeCleaned.AddLast(propPosition
    );
79.                 }
80.             }
81.
82.             foreach (var cleanTask in toBeCleaned) {

83.                 Clean(cleanTask);
84.             }
85.         }
86.
87.         private void OnSpawnFinished(Vector2Int pos)
     {
88.             if (referencePosition != null) {
89.                 Clean(referencePosition, referenceRa
    dius);
90.             }
91.         }
92.
93.         public void SetReferencePosition(Vector2Int
    position, int radius) {
94.             referencePosition = position;
95.             referenceRadius = radius;
96.         }
97.     }
98.
99.     /*** SOME CLASS ARE HIDDEN ***/
100. }
```

Kode Sumber 4.13 Implementasi kelas *PropSpawner*.

```
1.  using UnityEngine;
2.  using System.Collections.Generic;
3.  using System.Collections;
4.  using System;
```

```
5.  using Generative;
6.  using Rendoru.Collection;
7.  using Rendoru.Unity;
8.  using Rendoru.Scheduling;
9.
10. namespace TerrainGenerator {
11.
12.     public class SpawnPropTask : ITimedTask {
13.         private LinkedList<IDisposable> props;
14.         private Func<float, int, int, MonoDisposable
    GameObject> evalFunc;
15.
16.         private Action<Vector2Int> onFinishCallback;

17.         private NoiseMap noiseMap;
18.         private Vector3 distance;
19.
20.         private float density;
21.         private bool isFinished;
22.
23.
24.         public SpawnPropTask(
25.             LinkedList<IDisposable> props,
26.             NoiseMap noiseMap, Vector3 distance,
27.             float density,
28.             Func<float, int, int, MonoDisposableGame
    Object> evalFunc,
29.             Action<Vector2Int> onFinishCallback) {
30.
31.             isFinished = false;
32.             this.props = props;
33.             this.density = density;
34.             this.noiseMap = noiseMap;
35.             this.distance = distance;
36.             this.evalFunc = evalFunc;
37.             this.onFinishCallback = onFinishCallback
    ;
38.         }
39.
40.         public void Execute() {
41.             var resolution = noiseMap.Map.Length;
42.             var position = noiseMap.Position;
43.             var heightMap = noiseMap.Map;
```

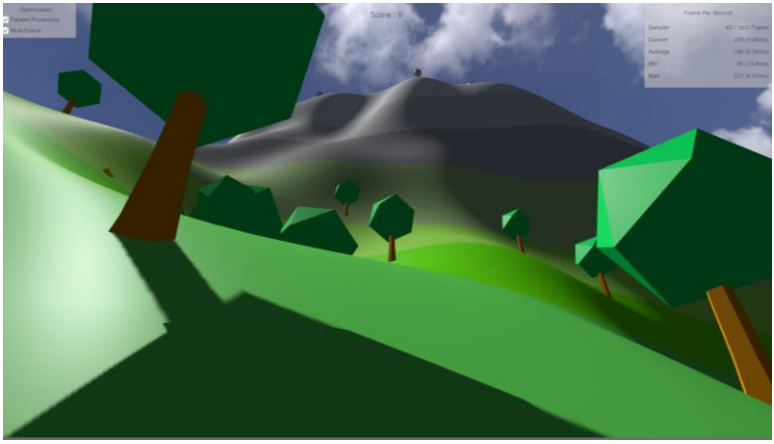
```
44.             var increment = (int)(resolution / densi
    ty);
45.             for (int i = 0; i < heightMap.Length; i
    += increment) {
46.                 for (int j = 0; j < heightMap[i].Len
    gth; j += increment) {
47.                     var value = heightMap[i][j];
48.                     if ((int)(value * 10000) % 2 > 0
    ) { //Give 1/2 chance to skip evaluation
49.                         for (int x = -
    1; x <= 1; x++) {
50.                             for (int y = -
    1; y <= 1; y++) {
51.                                 int xIdx = i + x;
52.                                 int yIdx = j + y;
53.                                 if (xIdx > 0 && xIdx
     < heightMap.Length) {
54.                                     if (yIdx > 0 &&
    yIdx < heightMap.Length) {
55.                                         value = Math
    f.Min(value, heightMap[xIdx][yIdx]);
56.                                     }
57.                                 }
58.                             }
59.                         }
60.                         var evalResult = evalFunc(va
    lue, i + position.x, j + position.y);
61.                         if (evalResult != null) {
62.                             float xPosition = positi
    on.x * distance.x + distance.x * i / resolution - di
    stance.x / 2f;
63.                             float yPosition = distan
    ce.y * value - distance.y / 2f;
64.                             float zPosition = positi
    on.y * distance.z + distance.z * j / resolution - di
    stance.z / 2f;
65.                             evalResult.Object.transf
    orm.position = new Vector3(xPosition, yPosition, zPo
    sition);
66.                             props.AddLast(evalResult
    );
67.                         }
68.                     }
```

```
69.                 }
70.             }
71.             isFinished = true;
72.             onFinishCallback(position);
73.         }
74.
75.         public bool IsFinished() {
76.             return isFinished;
77.         }
78.     }
79.
80.     /*** SOME CLASS ARE HIDDEN ***/
81. }
```

Kode Sumber 4.14 Implementasi kelas *SpawnPropTask*.

4.12 Implementasi InteractiveControl

Kelas ini bertugas untuk menerima *input* dari pengguna dan menggerakkan karakter. Pemain dapat berjalan maupun melompat ke tempat yang mereka inginkan. Fungsi *UpdateCharacterMove* digunakan untuk menerima *input* dan memproses *input* menjadi gerakan karakter. Fungsi *UpdateView* digunakan untuk melakukan pembaruan dari posisi yang pemain hadapi saat ini. Fungsi *Active* akan dipanggil jika karakter ini teraktivasi dan digunakan untuk menginisialisasi posisi karakter untuk pertama kalinya. Gambar 4.15 menunjukkan tampilan dari kontrol interaktif.



Gambar 4.15 Implementasi tampilan kontrol interaktif.

```
1. using UnityEngine;
2. using System.Collections;
3. using Rendoru.Unity;
4.
5. namespace Gameplay {
6.     public class InteractiveControl : MonoBehaviour,
    ICharacterController {
7.
8.         [SerializeField] private float speed;
9.         [SerializeField] private float maxSpeed;
10.         [SerializeField] private float jumpPower;
11.         [SerializeField] private float viewSpeed;
12.
13.         private CharacterController characterControl
    ler;
14.
15.         private float jumpGravity;
16.         private bool isMouseLocked;
17.
18.         private void Awake() {
19.             characterController = GetComponent<Chara
    cterController>();
20.             isMouseLocked = false;
```

```
21.         }
22.
23.         private void Update() {
24.             UpdateCharacterMove();
25.             UpdateView();
26.         }
27.
28.
29.         private void UpdateCharacterMove() {
30.             var horizontal = Input.GetAxis("Horizont
    al");
31.             var vertical = Input.GetAxis("Vertical")
    ;
32.             var right = transform.right;
33.             var forward = transform.forward;
34.
35.             if (Input.GetKeyDown(KeyCode.Space)) {
36.                 jumpGravity = jumpPower;
37.             }
38.             var horizontalSpeed = right * horizontal
    ;
39.             var forwardSpeed = forward * vertical;
40.             var moveSpeed = new Vector3(horizontalSp
    eed.x + forwardSpeed.x, 0, horizontalSpeed.z + forwa
    rdSpeed.z) * speed * Time.deltaTime;
41.             var velocity2D = new Vector2(moveSpeed.x
    , moveSpeed.z);
42.             var xSpeed = moveSpeed.x;
43.             var zSpeed = moveSpeed.z;
44.             if (velocity2D.sqrMagnitude > maxSpeed *
     maxSpeed) {
45.                 var magnitude = velocity2D.magnitude
    ;
46.                 var multiplier = maxSpeed / magnitud
    e;
47.                 xSpeed = moveSpeed.x * multiplier;
48.                 zSpeed = moveSpeed.z * multiplier;
49.             }
50.             moveSpeed = new Vector3(xSpeed, jumpGrav
    ity, zSpeed);
51.             if (jumpGravity > 0f) {
52.                 jumpGravity = Mathf.Clamp(jumpGravit
    y - Time.deltaTime * jumpPower, 0, jumpPower);
```

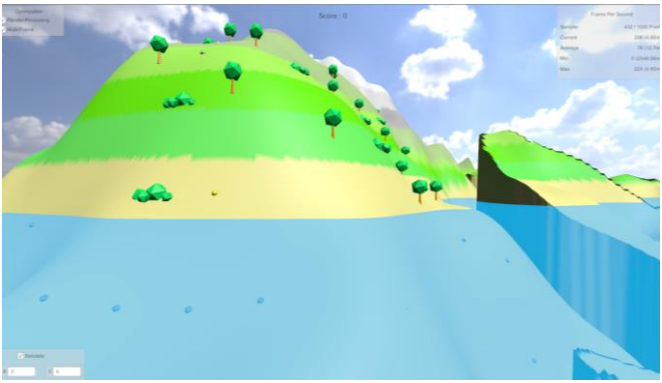
```
53.                     characterController.Move(Vector3.up
    * jumpGravity);
54.                 }
55.                 characterController.SimpleMove(moveSpeed
    );
56.             }
57.
58.             private void UpdateView() {
59.                 if (Input.GetKeyDown(KeyCode.LeftAlt)) {

60.                     isMouseLocked = !isMouseLocked;
61.                     if (isMouseLocked) {
62.                         Cursor.lockState = CursorLockMod
    e.Locked;
63.                     }
64.                     else {
65.                         Cursor.lockState = CursorLockMod
    e.None;
66.                     }
67.                 }
68.                 var mouseX = Input.GetAxis("Mouse X");
69.                 var mouseY = Input.GetAxis("Mouse Y");
70.                 var localAngle = transform.localEulerAng
    les;
71.                 if (localAngle.x < 0) {
72.                     localAngle.x += 360;
73.                 }
74.                 if (localAngle.y < 0) {
75.                     localAngle.y += 360;
76.                 }
77.
78.                 if (localAngle.x > 0) {
79.                     localAngle.x -= 360;
80.                 }
81.                 if (localAngle.y > 0) {
82.                     localAngle.y -= 360;
83.                 }
84.                 localAngle.x = (localAngle.x - mouseY *
    viewSpeed * Time.deltaTime);
85.                 localAngle.y = (localAngle.y + mouseX *
    viewSpeed * Time.deltaTime);
86.                 transform.localEulerAngles = localAngle;
```

```
87.         }
88.
89.         public void Activate(bool value, Transform c
    ameraTransform) {
90.             if (value) {
91.                 var bodyTransform = RenBehaviour.Tra
    ceComponent<Transform>(transform, "Body");
92.                 cameraTransform.SetParent(bodyTransf
    orm);
93.                 cameraTransform.localPosition = Vect
    or3.zero;
94.
95.                 RaycastHit hit;
96.                 var rayOrigin = new Vector3(transfor
    m.position.x, 10000, transform.position.z);
97.                 if (Physics.Raycast(rayOrigin, Vecto
    r3.down, out hit)) {
98.                     if (hit.collider != null) {
99.                         transform.position = hit.poi
    nt + Vector3.up * 2;
100.                    }
101.                }
102.                Cursor.lockState = CursorLockMode.Lo
    cked;
103.                isMouseLocked = true;
104.            }
105.            else {
106.                Cursor.lockState = CursorLockMode.No
    ne;
107.                isMouseLocked = false;
108.            }
109.            gameObject.SetActive(value);
110.        }
111.
112.        public Transform GetTransform() {
113.            return transform;
114.        }
115.    }
116. }
```

Kode Sumber 4.15 Implementasi kelas *InteractiveControl*.

4.13 Implementasi AutomaticControl

Kelas ini bertugas untuk menyimulasikan pergerakan ke arah tertentu dengan kecepatan yang sama. Kelas ini digunakan untuk melakukan pemantauan terhadap performa pembuatan saat ini. Fungsi *Update* akan membuat karakter ini berjalan ke arah tertentu. Fungsi *UpdateAngle* akan mengarahkan kamera ke arah yang sedang karakter ini tuju. Gambar 4.16 menunjukkan tampilan dari kontrol otomatis.



Gambar 4.16 Implementasi tampilan kontrol otomatis.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using Rendoru.Unity;
4.
5.  namespace Gameplay {
6.      public class AutomaticControl : MonoBehaviour, ICharacterCont
    roller {
7.
8.          private Transform body;
9.          private CameraWalker cameraWalker;
10.
11.         private Vector3 lastPosition;
12.
13.
```

```
14.     private void Awake() {
15.         body = RenBehaviour.TraceComponent<Transform>(transfo
    rm, "Body");
16.         cameraWalker = GetComponent<CameraWalker>();
17.         lastPosition = transform.position;
18.     }
19.
20.     private void Update() {
21.         UpdateAngle();
22.     }
23.     public void Activate(bool value, Transform cameraTransform)
    {
24.         gameObject.SetActive(value);
25.         if (value) {
26.             cameraTransform.SetParent(body);
27.             cameraWalker.SetVelocity(new Vector2(0, 100));
28.         }
29.     }
30.
31.     public Transform GetTransform() {
32.         return transform;
33.     }
34.
35.     private void UpdateAngle() {
36.         var delta = transform.position - lastPosition;
37.         delta.Normalize();
38.         var targetRotation = Quaternion.LookRotation(delta, Vector3
    .up);
39.         var originalEuler = transform.rotation.eulerAngles;
40.         var diff = originalEuler - targetRotation.eulerAngles;
41.         var quartenionLerp = Quaternion.Lerp(transform.rotation, ta
    rgetRotation, Time.deltaTime);
42.         transform.rotation = quartenionLerp;
43.         lastPosition = transform.position;
44.     }
45.   }
46. }
```

Kode Sumber 4.16 Implementasi kelas *AutomaticControl*.

4.14 Implementasi PerformancePanel

Kelas ini bertugas untuk menampilkan performa permainan saat ini. Kelas ini bekerja dengan cara mengambil sampel tiap *frame* dan setiap 1000 *frame* akan memperbaharui tampilan panel performa saat ini. Adapun data yang ditampilkan adalah rata – rata *frame rate* per 1000 *frame*, waktu maksimal dan minimal pemrosesan per 1000 *frame*. Kelas ini digunakan untuk mengambil data untuk uji coba dan evaluasi. Gambar 4.17 menunjukkan tampilan panel ini.

Gambar 4.17 Implementasi tampilan panel performa.

```
1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. using UnityEngine.UI;
5. using Rendoru.Unity;
6.
7. namespace Gameplay {
8.     public class PerformancePanel : MonoBehaviour {
```

```
9.
10.     [SerializeField] private int samplingFrame = 1000;
11.
12.
13.     private Text currentText;
14.     private Text averageText;
15.     private Text minText;
16.     private Text maxText;
17.     private Text samplerText;
18.
19.     private float sampleTotal;
20.     private float minSample = 9999;
21.     private float maxSample = -1;
22.
23.     private int frame = 0;
24.
25.     private void Awake() {
26.         var valueTransform = RenBehaviour.TraceComponent<Trans
form>(transform, "Panel/Value");
27.         currentText = RenBehaviour.TraceComponent<Text>(valueT
ransform, "Current");
28.         averageText = RenBehaviour.TraceComponent<Text>(value
Transform, "Average");
29.         minText = RenBehaviour.TraceComponent<Text>(valueTran
sform, "Min");
30.         maxText = RenBehaviour.TraceComponent<Text>(valueTran
sform, "Max");
31.
32.         samplerText = RenBehaviour.TraceComponent<Text>(value
Transform, "Sample");
33.
34.     }
35.     private void Update() {
36.         var timeDelta = Time.unscaledDeltaTime * 1000;
37.         currentText.text = ToSampleText(timeDelta);
38.         samplerText.text = string.Format("{0} / {1} Frame", frame, s
amplingFrame);
39.         UpdateSample();
40.         if (frame == samplingFrame) {
41.             UpdateSampleView();
42.         }
43.     }
```

```
44.
45.     private void UpdateSample() {
46.         var timeDelta = Time.unscaledDeltaTime * 1000;
47.         frame++;
48.         sampleTotal += timeDelta;
49.         if (minSample > timeDelta) {
50.             minSample = timeDelta;
51.         }
52.         if (maxSample < timeDelta) {
53.             maxSample = timeDelta;
54.         }
55.     }
56.
57.     private void UpdateSampleView() {
58.         averageText.text = ToSampleText(sampleTotal / samplingFr
    ame);
59.         minText.text = ToSampleText(maxSample);
60.         maxText.text = ToSampleText(minSample);
61.         frame = 0;
62.         minSample = 99999;
63.         maxSample = -1;
64.         sampleTotal = 0;
65.     }
66.
67.     private string ToSampleText(float timeDelta) {
68.         var currentFPS = 1000 / timeDelta;
69.         return string.Format("{0} ({1:0.##}ms)", (int)currentFPS, ti
    meDelta);
70.     }
71.   }
72. }
```

Kode Sumber 4.17 Implementasi kelas *PerformancePanel*.

4.15 Implementasi TimedTaskRunner

Kelas ini bertugas untuk melakukan eksekusi terbatas antar *frame*. Fungsi *Execute* akan dipanggil pada setiap *frame* dan membatasi waktu yang diperbolehkan untuk memproses tugas. Hal ini memungkinkan untuk mengeksekusi tugas di banyak *frame*. Semua tugas yang masuk dikerjakan secara *first in first out* sehingga tidak menyebabkan *starvation*. Semua tugas yang akan diproses harus mengimplementasi *interface ITimedTask* untuk dapat di proses.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System.Diagnostics;
4.  using System.Collections.Generic;
5.  using System;
6.  using System.Threading;
7.
8.
9.  namespace Rendoru.Scheduling {
10.
11.   /// <summary>
12.   /// Capable of doing work on limited time constraint, may overfl
    ow execution time when task is too big
13.   /// Thread Safe
14.   /// </summary>
15.   public class TimedTaskRunner : ITimedTaskRunner {
16.
17.     private int maxDuration;
18.     private Queue<ITimedTask> tasks;
19.     private Mutex mutex;
20.
21.     public static bool IsTimedRun = true; //TODO [0001] : Unsafe
    please remove, for demonstration purposes only
22.     public TimedTaskRunner(int maxDuration) {
23.       this.maxDuration = maxDuration;
24.       tasks = new Queue<ITimedTask>();
25.       mutex = new Mutex();
26.     }
27.
28.     public void AddTask(ITimedTask task) {
```

```
29.
30.         if (IsTimedRun) {
31.             mutex.WaitOne();
32.             tasks.Enqueue(task);
33.             mutex.ReleaseMutex();
34.         }
35.         else {
36.             task.Execute(); //TODO [0001] : Unsafe
37.         }
38.     }
39.
40.     public void Execute() {
41.         var stopwatch = new Stopwatch();
42.         stopwatch.Start();
43.         var task = Peek();
44.         while (task != null) {
45.             if (task.IsFinished()) {
46.                 task = GetNext();
47.             }
48.             else {
49.                 task.Execute(); //Delayed so that if task doesn't need to
    be worked will be skipped
50.             }
51.             if (IsTimedRun) { //TODO [0001] : Unsafe
52.                 var milis = stopwatch.ElapsedMilliseconds;
53.                 if (milis > maxDuration) {
54.                     break;
55.                 }
56.             }
57.         }
58.         stopwatch.Stop();
59.     }
60.
61.     protected ITimedTask Peek() {
62.         mutex.WaitOne();
63.         ITimedTask task = null;
64.         if (tasks.Count > 0) {
65.             task = tasks.Peek();
66.         }
67.         mutex.ReleaseMutex();
68.         return task;
69.     }
```

```
70.
71.     protected ITimedTask GetNext() {
72.         mutex.WaitOne();
73.         ITimedTask task = null;
74.         tasks.Dequeue();
75.         if (tasks.Count > 0) {
76.             task = tasks.Peek();
77.         }
78.         mutex.ReleaseMutex();
79.         return task;
80.     }
81.
82.   }
83. }
```

Kode Sumber 4.18 Implementasi kelas pemrosesan *TimedTaskRunner*.

4.16 Implementasi ParallelRunner

Kelas ini bertugas untuk melakukan eksekusi secara paralel dengan batasan prosesor. Semua tugas yang masuk di sini akan dikerjakan secara paralel, namun jika semua *thread* yang disimpan dalam keadaan sibuk maka, akan diantrekan untuk diproses oleh *thread* yang sedang tidak sibuk. Seluruh tugas yang masuk di kelas ini harus mengimplementasikan *interface ITask*.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using System.Threading;
5.  using System;
6.
7.  namespace Rendoru.Scheduling {
8.
9.    /// <summary>
10.   /// Capable of processing task in parallel, a task must be thread s
    afe to operate
```

```
11.   /// Will limit processing core to a maximum of MAX_PROCESSOR
    - 2 to provide stability
12.   /// </summary>
13.   public class ParallelRunner : ITaskRunner {
14.
15.     private static ParallelRunner globalInstance;
16.
17.     private Queue<ITask> tasks;
18.     private Mutex mutex;
19.     private int workerCount = 0;
20.     private int maxThread = 0;
21.
22.     public static bool IsMultiThreaded = true; //TODO [0001] : U
    nsafe Please Remove, it's for demonstration purposes only
23.
24.
25.     public ParallelRunner() {
26.       mutex = new Mutex();
27.       tasks = new Queue<ITask>();
28.       maxThread = Environment.ProcessorCount - 2;
29.       if (maxThread <= 0) {
30.         maxThread = 1;
31.       }
32.     }
33.
34.     public void AddTask(ITask task) {
35.       if (IsMultiThreaded) { //TODO [0001] : Unsafe
36.         mutex.WaitOne();
37.         tasks.Enqueue(task);
38.         if (workerCount < maxThread) {
39.           workerCount++;
40.           ThreadPool.QueueUserWorkItem(Execute);
41.         }
42.         mutex.ReleaseMutex();
43.       }
44.       else {
45.         task.Execute(); //TODO [0001] : Unsafe
46.       }
47.     }
48.
49.     private void Execute(object state) {
50.       while (true) {
```

```
51.             mutex.WaitOne();
52.             if (tasks.Count <= 0) {
53.                 workerCount--;
54.                 mutex.ReleaseMutex();
55.                 break;
56.             }
57.             var task = tasks.Dequeue();
58.             mutex.ReleaseMutex();
59.             try {
60.                 task.Execute();
61.             }
62.             catch (Exception e) {
63.                 Debug.LogError(e);
64.             }
65.         }
66.     }
67.
68.     /// <summary>
69.     /// Return a global instance of ParallelRunner
70.     /// </summary>
71.     /// <returns></returns>
72.     public static ParallelRunner GetGlobalInstance() {
73.         if (globalInstance == null) {
74.             globalInstance = new ParallelRunner();
75.         }
76.         return globalInstance;
77.     }
78. }
79. }
```

Kode Sumber 4.19 Implementasi kelas pemrosesan *ParallelRunner*.

[*Halaman ini sengaja dikosongkan*]

BAB V
UJICOBA DAN EVALUASI

Bab ini akan membahas tentang uji coba dan evaluasi dari sistem yang telah dirancang dan dibuat. Uji coba ini dilakukan untuk mengetahui kinerja sistem dengan lingkungan uji coba yang telah ditentukan.

5.1 Lingkungan Uji Coba

Lingkungan pengujian dari terdari 2 macam perangkat keras yaitu *personal computer* dan juga *smartphone*. Hal ini bertujuan untuk mendapatkan hasil dari berbagai macam platform dan mengetahui efektivitas dari pendekatan yang digunakan.

Untuk pengujian dengan *personal computer (PC)* menggunakan perangkat *Dell* 7567. Untuk perangkat *Dell* 7567, sistem operasi yang digunakan adalah *Windows* 10 64-bit. PC yang digunakan memiliki spesifikasi perangkat keras dengan prosesor *Intel* i7 7700HQ dengan kecepatan 2.8 GHz, *Random Access Memory (RAM)* sebesar 16 GB, dan mempunyai *Graphics Processing Unit (GPU)* yaitu *NVIDIA GeForce* GTX 1050TI dengan *VRAM* sebesar 4 GB. Tampilan resolusi yang digunakan adalah 1920 x 1080 *pixel*.

Untuk pengujian dengan *smartphone* digunakanlah 2 macam *smartphone*. Yaitu *Xiaomi Redmi* 9T Pro dengan spesifikasi resolusi layar 1080 x 2340 *pixel*. Sistem operasi yang digunakan adalah *Android* 10. Prosesor yang digunakan adalah *Qualcomm Snapdragon* 855 *Octa-core* (1 x 2.84 GHz Kryo 485 & 3 x 2.42 GHz Kryo 485 & 4 x 1.78 GHz Kryo 485). *Graphic Processing Unit (GPU)* yang digunakan adalah *Adreno* 640. *Ukuran Random Access Memory (RAM)* sebesar 6 GB. *Smartphone* kedua yang digunakan adalah *Xiaomi Redmi* 4X dengan spesifikasi resolusi layar 720 x 1080 *pixel*. Sistem operasi yang digunakan adalah *Android* 7.1.2 (*Nougat*). Prosesor yang digunakan adalah *Qualcomm Snapdragon* 435 *Octa-core* 1.4 GHz *Cortex*-A53.

Graphic Processing Unit (GPU) yang digunakan adalah *Adreno* 505. Ukuran *Random Access Memory (RAM)* sebesar 2 GB.

Pengujian tambahan juga dilakukan menggunakan perangkat *Acer* E5 553G. Untuk perangkat *Acer* E5 553G, sistem operasi yang digunakan adalah *Windows* 10 64-bit. PC yang digunakan memiliki spesifikasi perangkat keras dengan prosesor *AMD* FX 9800P dengan kecepatan 2.7 GHz, *Random Access Memory (RAM)* sebesar 8 GB, dan mempunyai *Graphics Processing Unit (GPU)* yaitu *Radeon* R8 M445DX dengan *VRAM* sebesar 2 GB. Tampilan resolusi yang digunakan adalah 1366 x 768 *pixel*. Untuk perangkat *smartphone* dilakukan uji coba pada perangkat *Oppo* A57 dan juga *Vivo* Y12. Untuk perangkat *Oppo* A57 memiliki spesifikasi yaitu sistem operasi yang digunakan adalah *Android* 10. Prosesor yang digunakan adalah *Qualcomm* MSM8940 *Snapdragon* 435 *Octa-core* 1.4 GHz *Cortex*-A53. *Graphic Processing Unit (GPU)* yang digunakan adalah *Adreno* 505 dengan resolusi layar sebesar 720 x 1280 *pixel*. *Ukuran Random Access Memory (RAM)* sebesar 3GB. Sedangkan untuk perangkat *Vivo* Y12 memiliki spesifikasi yaitu sistem operasi yang digunakan adalah *Android* 9. Prosesor yang digunakan adalah *Mediatek* MT6762 *Helio* P22 dengan *Octa-core* 2.0 GHz *Cortex*-A53. *Graphic Processing Unit (GPU)* yang digunakan adalah *PowerVR* GE8320 dengan resolusi layar sebesar 720 x 1544 *pixel*. *Ukuran Random Access Memory (RAM)* sebesar 3 GB.

5.2 Skenario Pengujian

Pengujian dilakukan dengan cara menguji coba perangkat pengujian dengan perbedaan parameter. Pengujian akan dilakukan dengan cara menjalankan mode simulasi dan memantau hasilnya dengan menggunakan panel performa yang ada di dalam permainan dan juga memantau performa saat ini menggunakan *profiler* yang berada di *Unity*. Penggunaan *profiler* dipilih karena dapat diambil detail per-*frame* yang sedang di proses. Pengujian akan dilakukan dengan menggunakan aplikasi yang sudah dibuat

menjadi sebuah aplikasi *executable* agar tidak tercampuri dengan *overhead* yang dihasilkan oleh *Unity*.

Tidak semua perangkat dapat menjalankan parameter yang sama. Seperti saja *simplex noise* yang saat ini hanya didukung untuk perangkat berupa *personal computer*. Untuk pengujian antar perangkat pada parameter tersebut digunakanlah parameter yang sama yaitu menggunakan *perlin noise*. Namun untuk pengujian menggunakan perangkat berupa *personal computer* tetap diakan 2 macam pengujian yaitu menggunakan *perlin noise* dan *simplex noise* serta membandingkan kedua hasilnya. Tabel 5.1 menunjukkan parameter yang digunakan.

Tabel 5.1 Parameter pada skenario uji coba.

Nama	Nilai
Kendali	Otomatis
Kecepatan bergerak	100
Ukuran petak (panjang, lebar, dan tinggi)	500 x 500 x 2500
Resolusi petak	200
Tahap resolusi	3, 1, 1, 1, 4
Densitas objek terbuat	10
Ukuran *collider* yang dituju	50
Metode pengambilan sampel *noise*	*Simplex* (*PC*) dan *Perlin* (*PC* dan *smartphone*)
Skala pengambilan sampel	5 untuk *Simplex* dan 0.25 untuk *Perlin*
Faktor H pada *FBM*	1
Jumlah Oktaf pada *FBM*	5
Resolusi material yang dituju	50

5.3 Hasil Uji Coba dan Evaluasi

Pada pengujian menggunakan perangkat berupa *Personal Computer (PC)* dilakukan sebanyak 2 kali. Yaitu menggunakan *simplex noise* dan *perlin noise*.

Tabel 5.2 menunjukkan hasil setelah menggunakan metode yang diajukan. Dengan parameter yang cukup berat yaitu dengan resolusi sebesar 200 (200 x 200 unit data per petak), sistem dapat memberikan waktu pemrosesan yang cukup baik di mana waktu maksimum pemrosesan pada saat terjadi perubahan berada pada angka 29.87*ms*. Sedangkan waktu minimum ketika terjadi pemrosesan perubahan tercetak sebesar 18ms. Jika kita bandingkan dengan waktu yang dibutuhkan untuk memproses perubahan tanpa menggunakan metode yang diajukan maka kita dapat melihat hasilnya yang berkisar antara 210ms hingga 243ms yang ditunjukkan oleh tabel 5.5. Hal ini menunjukkan peningkatan sebesar 8 hingga 11 kali dengan menggunakan metode yang diajukan. Jika kita bandingkan ketika menggunakan salah satu metode yang diajukan didapati bahwa pemrosesan secara paralel memberikan dampak tertinggi yaitu sebesar 7 kali lipat waktu terlama dengan seluruh metode yang digunakan seperti yang ditunjukkan oleh tabel 5.4. Sedangkan berdasarkan tabel 5.3 didapati bahwa pemrosesan antar *frame* menyebabkan peningkatan waktu pemrosesan sebesar 50ms atau 1.7 kali lipat.

Tabel 5.2 Pengujian perangkat *Dell* 7567 dengan *perlin noise.*

No	*Average*	*Min*	*Max*
2000	60 (16.67ms)	33 (29.87ms)	400 (2.5ms)
3000	60 (16.67ms)	50 (19.77ms)	79 (12.61ms)
4000	59 (16.67ms)	47 (21.12ms)	90 (11.03ms)
5000	60 (16.67ms)	49 (20.22ms)	79 (12.52ms)
6000	60 (16.67ms)	55 (18ms)	69 (14.33ms)

Tabel 5.3 Pengujian perangkat *Dell* 7567 dengan *perlin noise* dan tanpa pemrosesan antar *frame*.

No	*Average*	*Min*	*Max*
2000	59 (16.82ms)	15 (64.64ms)	248 (4.02ms)
3000	59 (16.83ms)	17 (58.51ms)	225 (4.42ms)
4000	59 (16.88ms)	18 (55.48ms)	156 (6.38ms)
5000	59 (16.83ms)	17 (56.12ms)	175 (5.68ms)
6000	59 (16.85ms)	19 (50.73ms)	151 (6.59ms)

Tabel 5.4 Pengujian perangkat *Dell* 7567 dengan *perlin noise* dan tanpa pemrosesan paralel.

No	*Average*	*Min*	*Max*
2000	58 (17.18ms)	4 (213.38ms)	141 (7.05ms)
3000	57 (17.35ms)	4 (213.97ms)	155 (6.45ms)
4000	57 (17.25ms)	4 (242.68ms)	148 (6.75ms)
5000	57 (17.38ms)	4 (234.59ms)	162 (6.14ms)
6000	58 (17.23ms)	3 (258.85ms)	174 (5.73ms)

Tabel 5.5 Pengujian perangkat *Dell* 7567 dengan *perlin noise* dan tanpa optimasi pemrosesan.

No	*Average*	*Min*	*Max*
2000	57 (17.45ms)	4 (241.9ms)	261 (3.82ms)
3000	56 (17.73ms)	5 (215.5ms)	136 (7.31ms)
4000	57 (17.45ms)	4 (210.85ms)	126 (7.92ms)
5000	57 (17.5ms)	4 (243.79ms)	284 (3.52ms)
6000	56 (17.68ms)	4 (218.11ms)	165 (6.04ms)

Tabel 5.6 menunjukkan pengujian menggunakan *simplex noise*. Jika kita bandingkan dengan menggunakan metode *perlin noise*, tidak terjadi perubahan yang signifikan ketika menggunakan

metode yang diajukan. Namun perubahan dapat terlihat ketika pemrosesan secara paralel di nonaktifkan seperti yang ditunjukkan oleh tabel 5.8. Dengan membandingkan hasil dari tabel 5.5 dengan tabel 5.8 maka terjadi peningkatan waktu pemrosesan sebesar 54ms. Hal ini menunjukkan bahwa jika kita menggunakan pemrosesan secara paralel maka kita diberi kebebasan untuk menggunakan metode yang menurut kita lebih baik.

Tabel 5.6 Pengujian perangkat *Dell* 7567 dengan *simplex noise*.

No	*Average*	*Min*	*Max*
2000	59 (16.67ms)	30 (32.75ms)	130 (7.66ms)
3000	60 (16.67ms)	51 (19.25ms)	70 (14.15ms)
4000	60 (16.67ms)	42 (23.44ms)	101 (9.85ms)
5000	60 (16.67ms)	53 (18.73ms)	67 (14.71ms)
6000	60 (16.67ms)	50 (19.89ms)	72 (13.78ms)

Tabel 5.7 Pengujian perangkat *Dell* 7567 dengan *simplex noise* dan tanpa pemrosesan antar *frame.*

No	*Average*	*Min*	*Max*
2000	58 (17ms)	9 (100.16ms)	260 (3.84ms)
3000	59 (16.88ms)	15 (66.19ms)	95 (10.51ms)
4000	59 (16.85ms)	17 (56.27ms)	98 (10.1ms)
5000	59 (16.92ms)	16 (60.59ms)	95 (10.51ms)
6000	59 (16.87ms)	16 (58.83ms)	76 (13.05ms)

Tabel 5.8 Pengujian perangkat *Dell* 7567 dengan *simplex noise* dan tanpa pemrosesan paralel.

No	*Average*	*Min*	*Max*
2000	57 (17.47ms)	3 (312.49ms)	144 (6.92ms)
3000	57 (17.38ms)	3 (281.59ms)	138 (7.24ms)
4000	57 (17.53ms)	3 (262.26ms)	179 (5.58ms)
5000	57 (17.28ms)	3 (254.8ms)	117 (8.54ms)
6000	56 (17.58ms)	3 (276.29ms)	156 (6.38ms)

Tabel 5.9 Pengujian perangkat *Dell* 7567 dengan *simplex noise* dan tanpa optimasi pemrosesan.

No	*Average*	*Min*	*Max*
2000	55 (17.97ms)	2 (349.81ms)	124 (8.01ms)
3000	55 (18.07ms)	3 (301.19ms)	257 (3.88ms)
4000	56 (17.62ms)	3 (265.44ms)	225 (4.43ms)
5000	55 (17.87ms)	3 (273.44ms)	192 (5.18ms)
6000	56 (17.58ms)	3 (270.97ms)	134 (7.42ms)

Tabel 5.10 menunjukkan pengujian menggunakan *perlin noise* di perangkat *smartphone* yang berperforma tinggi. Jika kita bandingkan dengan hasil dari tabel 5.2 terlihat bahwa terjadi perbedaan yang cukup jauh. Hal ini menandakan bahwa parameter yang digunakan harus disesuaikan dengan perangkat yang dituju. Dengan parameter yang sama, waktu yang dibutuhkan untuk memproses meskipun sudah menggunakan metode yang diajukan masih cukup tinggi yaitu 74ms. Jika kita bandingkan tabel 5.13 dengan tabel 5.5 terlihat waktu yang hampir sama pada perbandingan hasil pada perangkat *PC* yaitu 7 hingga 10 kali lipat jika tanpa menggunakan metode yang diajukan. Tabel 5.12 menunjukkan hasil yang sama di mana pemrosesan paralel tetap memberikan kontribusi terbaik dalam pemrosesan. Namun pada perangkat ini terjadi peningkatan yang cukup signifikan jika tanpa menggunakan pemrosesan antar *frame* yaitu hingga 2.9 kali lipat seperti yang ditunjukkan oleh tabel 5.11.

Tabel 5.10 Pengujian perangkat *Xiaomi* 9T Pro dengan *perlin noise*.

No	*Average*	*Min*	*Max*
2000	55 (17.96ms)	18 (54.15ms)	155 (6.44ms)
3000	56 (17.78ms)	22 (45.38ms)	128 (7.77ms)
4000	55 (18.1ms)	13 (74.36ms)	155 (6.44ms)
5000	56 (17.55ms)	20 (47.8ms)	174 (5.72ms)
6000	53 (18.58ms)	19 (52.12ms)	129 (7.72ms)

Tabel 5.11 Pengujian perangkat *Xiaomi* 9T Pro dengan *perlin noise* dan tanpa pemrosesan antar *frame*.

No	*Average*	*Min*	*Max*
2000	50 (19.73ms)	4 (221.78ms)	147 (6.76ms)
3000	48 (20.41ms)	3 (307.34ms)	92 (10.79ms)
4000	48 (20.55ms)	3 (296.23ms)	91 (10.96ms)
5000	46 (21.3ms)	3 (288.84ms)	141 (7.06ms)
6000	47 (21.03ms)	3 (287.76ms)	121 (8.26ms)

Tabel 5.12 Pengujian perangkat *Xiaomi* 9T Pro dengan *perlin noise* dan tanpa pemrosesan paralel.

No	*Average*	*Min*	*Max*
2000	53 (18.78ms)	2 (383.25ms)	134 (7.43ms)
3000	51 (19.35ms)	2 (354.06ms)	116 (8.6ms)
4000	51 (19.58ms)	2 (464.62ms)	122 (8.17ms)
5000	47 (20.87ms)	1 (533.25ms)	127 (7.87ms)
6000	48 (20.56ms)	2 (464.65ms)	170 (5.87ms)

Tabel 5.13 Pengujian perangkat *Xiaomi* 9T Pro dengan *perlin noise* dan tanpa optimasi pemrosesan.

No	*Average*	*Min*	*Max*
2000	49 (20.27ms)	1 (524.77ms)	129 (7.69ms)
3000	45 (21.89ms)	2 (471.6ms)	97 (10.21ms)
4000	41 (24.07ms)	1 (516.99ms)	81 (12.25ms)
5000	43 (23.06ms)	1 (526.29ms)	121 (8.26ms)
6000	43 (23.23ms)	1 (567.87ms)	154 (6.46ms)

Perangkat lain yang diuji coba adalah smartphone Xiaomi 4X. Perangkat ini termasuk perangkat yang tidak mempunyai performa yang cukup bagus untuk memproses hal berat dikarenakan perangkat keras yang dibawanya. Tabel 5.14 menunjukkan bahwa perangkat ini tidak mampu memproses

pekerjaan yang cukup berat dengan waktu pemrosesan hingga 381ms meskipun sudah menerapkan metode yang diajukan. Namun terlihat peningkatan yang signifikan jika dibandingan dengan tabel 5.17 yang jika tanpa menggunakan metode yang diajukan membutuhkan waktu pemrosesan hingga 2488ms atau 6.5 kali lipat. Tabel 5.15 tetap menunjukkan peristiwa yang sama dimana pemrosesan parallel memberikan dampak paling tinggi.

Tabel 5.14 Pengujian perangkat *Xiaomi* 4X dengan *perlin noise*.

No	*Average*	*Min*	*Max*
2000	27 (36.93ms)	2 (381.45ms)	63 (15.36ms)
3000	27 (36.89ms)	5 (194.3ms)	88 (11.26ms)
4000	28 (35.12ms)	6 (157.69ms)	98 (10.2ms)
5000	26 (37.88ms)	6 (165.42ms)	85 (11.7ms)
6000	27 (36.36ms)	5 (174.43ms)	91 (10.98ms)

Tabel 5.15 Pengujian perangkat *Xiaomi* 4X dengan *perlin noise* dan tanpa pemrosesan antar *frame*.

No	*Average*	*Min*	*Max*
2000	19 (50.81ms)	0 (1218.37ms)	64 (15.39ms)
3000	18 (53.48ms)	0 (1015.91ms)	64 (15.44ms)
4000	18 (53.09ms)	1 (943.43ms)	70 (14.26ms)
5000	18 (53.97ms)	1 (986.21ms)	66 (14.97ms)
6000	17 (56.77ms)	1 (968.89ms)	59 (16.71ms)

Tabel 5.16 Pengujian perangkat *Xiaomi* 4X dengan *perlin noise* dan tanpa pemrosesan paralel.

No	*Average*	*Min*	*Max*
2000	19 (51.95ms)	0 (2252.62ms)	60 (16.5ms)
3000	15 (62.97ms)	0 (2366.95ms)	56 (17.68ms)
4000	15 (64.07ms)	0 (2572.32ms)	62 (15.94ms)
5000	15 (64.07ms)	0 (2262.93ms)	83 (12.04ms)
6000	16 (60.79ms)	0 (2149.49ms)	78 (12.8ms)

Tabel 5.17 Pengujian perangkat *Xiaomi* 4X dengan *perlin noise* dan tanpa optimasi pemrosesan.

No	*Average*	*Min*	*Max*
2000	15 (64.05ms)	0 (2241.8ms)	74 (13.38ms)
3000	14 (67.16ms)	0 (2386.05ms)	56 (17.55ms)
4000	14 (68.91ms)	0 (2488.42ms)	70 (14.19ms)
5000	14 (69.01ms)	0 (2289.02ms)	59 (16.74ms)
6000	12 (82.06ms)	0 (2116.2ms)	54 (18.32ms)

Pada pengujian terhadap perangkat *Acer* E5 553G menunjukkan hasil yang serupa dengan perangkat *Dell* 7567 di mana hasil terbaik ditunjukkan oleh tabel 5.18 dengan menggunakan metode yang diajukan. Hal ini juga konsisten terhadap penggantian metode pengambilan sampel dari *perlin noise* menjadi *simplex noise* di mana terjadi penurunan performa jika tabel 5.18 dibandingkan dengan tabel 5.22.

Tabel 5.18 Pengujian perangkat *Acer* E5 553G dengan *perlin noise*.

No.	*Average*	*Min*	*Max*
2000	59 (16.88ms)	12 (81.17ms)	147 (6.76ms)
3000	58 (16.98ms)	17 (55.92ms)	204 (4.9ms)
4000	58 (17.03ms)	19 (51.15ms)	170 (5.87ms)
5000	58 (17.04ms)	21 (45.94ms)	220 (4.53ms)
6000	59 (16.88ms)	24 (40.32ms)	178 (5.6ms)

Tabel 5.19 Pengujian perangkat *Acer* E5 553G dengan *perlin noise* dan tanpa pemrosesan antar *frame*.

No.	*Average*	*Min*	*Max*
2000	57 (17.5ms)	5 (184.42ms)	245 (4.07ms)
3000	56 (17.58ms)	5 (176.3ms)	234 (4.27ms)
4000	55 (18.04ms)	5 (190.47ms)	262 (3.81ms)
5000	56 (17.61ms)	8 (116.6ms)	264 (3.78ms)
6000	54 (18.44ms)	6 (147.29ms)	218 (4.58ms)

Tabel 5.20 Pengujian perangkat *Acer* E5 553G dengan *perlin noise* dan tanpa pemrosesan paralel.

No.	*Average*	*Min*	*Max*
2000	54 (18.21ms)	2 (444.96ms)	180 (5.54ms)
3000	54 (18.33ms)	2 (354.83ms)	217 (4.59ms)
4000	54 (18.21ms)	2 (351.56ms)	153 (6.5ms)
5000	53 (18.54ms)	2 (419.71ms)	208 (4.8ms)
6000	54 (18.33ms)	2 (390.22ms)	205 (4.86ms)

Tabel 5.21 Pengujian perangkat *Acer* E5 553G dengan *perlin noise* dan tanpa optimasi pemrosesan.

No.	*Average*	*Min*	*Max*
2000	54 (18.51ms)	2 (383.82ms)	248 (4.02ms)
3000	51 (19.36ms)	2 (377.66ms)	191 (5.23ms)
4000	53 (18.6ms)	2 (401.96ms)	225 (4.44ms)
5000	52 (19.18ms)	2 (407.77ms)	230 (4.35ms)
6000	51 (19.41ms)	1 (507.66ms)	237 (4.22ms)

Tabel 5.22 Pengujian perangkat *Acer* E5 553G dengan *simplex noise*.

No.	*Average*	*Min*	*Max*
2000	58 (17.11ms)	9 (109.81ms)	264 (3.79ms)
3000	58 (17.21ms)	12 (79.8ms)	202 (4.94ms)
4000	57 (17.35ms)	12 (79.52ms)	254 (3.93ms)
5000	59 (16.88ms)	13 (75.55ms)	233 (4.28ms)
6000	57 (17.26ms)	11 (83.55ms)	194 (5.13ms)

Tabel 5.23 Pengujian perangkat *Acer* E5 553G dengan *simplex noise* dan tanpa pemrosesan antar *frame*.

No.	*Average*	*Min*	*Max*
2000	56 (17.83ms)	5 (198.14ms)	194 (5.15ms)
3000	57 (17.43ms)	5 (171.91ms)	363 (2.75ms)

4000	56 (17.54ms)	7 (126.64ms)	245 (4.08ms)
5000	57 (17.38ms)	8 (119.34ms)	359 (2.78ms)
6000	55 (18.11ms)	6 (156.46ms)	228 (4.37ms)

Tabel 5.24 Pengujian perangkat *Acer* E5 553G dengan *simplex noise* dan tanpa pemrosesan paralel.

No.	*Average*	*Min*	*Max*
2000	52 (18.99ms)	2 (480.62ms)	228 (4.38ms)
3000	54 (18.31ms)	2 (422.55ms)	239 (4.18ms)
4000	51 (19.24ms)	2 (431.02ms)	197 (5.06ms)
5000	53 (18.74ms)	2 (461.59ms)	178 (5.61ms)
6000	52 (18.93ms)	2 (441.88ms)	205 (4.87ms)

Tabel 5.25 Pengujian perangkat *Acer* E5 553G dengan *simplex noise* dan tanpa optimasi pemrosesan.

No.	*Average*	*Min*	*Max*
2000	52 (19.14ms)	1 (550.26ms)	278 (3.59ms)
3000	51 (19.51ms)	2 (465.17ms)	181 (5.51ms)
4000	52 (18.96ms)	2 (468.06ms)	247 (4.05ms)
5000	50 (19.93ms)	1 (501.58ms)	196 (5.1ms)
6000	49 (20.29ms)	2 (486.29ms)	237 (4.22 ms)

Pengujian pada perangkat *Oppo* A57 dan *Vivo* Y12 juga memberikan hasil yang sama dengan perangkat sebelumnya yaitu *Xiaomi* 9T Pro dan *Xiaomi* 4X di mana hasil terbaik didapat ketika menggunakan metode yang diajukan seperti yang ditunjukkan oleh tabel 5.26 dan tabel 5.30.

Tabel 5.26 Pengujian perangkat *Oppo* A57 dengan *perlin noise*.

No.	*Average*	*Min*	*Max*
2000	23 (41.97ms)	5 (166.96ms)	81 (12.34ms)

3000	24 (40.01ms)	6 (158.45ms)	123 (8.1ms)
4000	23 (42.13ms)	4 (202.74ms)	70 (14.2ms)
5000	24 (40.2ms)	5 (172.64ms)	83 (11.97ms)
6000	25 (38.94ms)	5 (181.27ms)	83 (11.93ms)

Tabel 5.27 Pengujian perangkat *Oppo* A57 dengan *perlin noise* dan tanpa pemrosesan antar *frame*.

No.	*Average*	*Min*	*Max*
2000	20 (48.62ms)	1 (782.09ms)	77 (12.88ms)
3000	19 (52.62ms)	1 (693.86ms)	70 (14.11ms)
4000	18 (53.22ms)	1 (677.91ms)	60 (16.59ms)
5000	19 (51.1ms)	1 (698.41ms)	75 (13.17ms)
6000	18 (53.18ms)	1 (732.05ms)	69 (14.35ms)

Tabel 5.28 Pengujian perangkat *Oppo* A57 dengan *perlin noise* dan tanpa pemrosesan paralel.

No.	*Average*	*Min*	*Max*
2000	19 (50.98ms)	0 (1395.95ms)	63 (15.84ms)
3000	19 (50.85ms)	0 (1432.73ms)	74 (13.47ms)
4000	19 (51.42ms)	0 (1525.45ms)	82 (12.16ms)
5000	18 (52.73ms)	0 (1378.17ms)	73 (13.57ms)
6000	20 (49.77ms)	0 (1300.48ms)	75 (13.2ms)

Tabel 5.29 Pengujian perangkat *Oppo* A57 dengan *perlin noise* dan tanpa optimasi pemrosesan.

No.	*Average*	*Min*	*Max*
2000	17 (56.51ms)	0 (1321.55ms)	72 (13.74ms)
3000	16 (60.31ms)	0 (1406.62ms)	69 (14.4ms)
4000	15 (63.16ms)	0 (1497.13ms)	72 (13.82ms)
5000	16 (62.19ms)	0 (1304.93ms)	69 (14.36ms)
6000	16 (59.39ms)	0 (1289.29ms)	70 (14.25ms)

Tabel 5.30 Pengujian perangkat *Vivo* Y12 dengan *perlin noise*.

No.	*Average*	*Min*	*Max*
2000	29 (33.84ms)	10 (93.98ms)	81 (12.29ms)
3000	29 (34.38ms)	11 (86.15ms)	84 (11.78ms)
4000	28 (35.03ms)	6 (159.92ms)	86 (11.61ms)
5000	28 (34.66ms)	10 (94.74ms)	72 (13.88ms)
6000	29 (34.45ms)	7 (131.88ms)	77 (12.89ms)

Tabel 5.31 Pengujian perangkat *Vivo* Y12 dengan *perlin noise* dan tanpa pemrosesan antar *frame*.

No.	*Average*	*Min*	*Max*
2000	25 (39.21ms)	2 (437.23ms)	82 (12.19ms)
3000	24 (40.83ms)	2 (413.9ms)	78 (12.7ms)
4000	25 (39.58ms)	2 (409.73ms)	78 (12.73ms)
5000	24 (40.59ms)	2 (399.21ms)	87 (11.42ms)
6000	25 (39.43ms)	2 (424.08ms)	81 (12.3ms)

Tabel 5.32 Pengujian perangkat *Vivo* Y12 dengan *perlin noise* dan tanpa pemrosesan paralel.

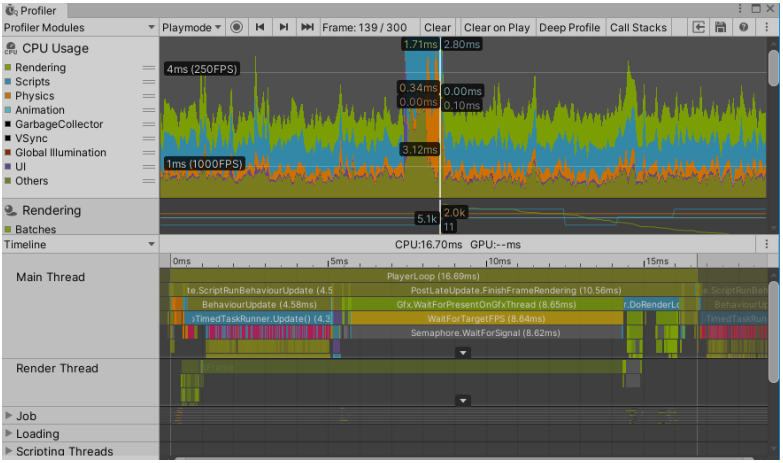
No.	*Average*	*Min*	*Max*
2000	25 (39.11ms)	0 (1089.64ms)	86 (11.55ms)
3000	25 (38.68ms)	0 (1008.41ms)	87 (11.46ms)
4000	27 (36.95ms)	0 (1074.09ms)	86 (11.57ms)
5000	24 (40.01ms)	0 (1042.42ms)	78 (12.8ms)
6000	25 (39.88ms)	1 (945.01ms)	76 (13.14ms)

Tabel 5.33 Pengujian perangkat *Vivo* Y12 dengan *perlin noise* dan tanpa optimasi pemrosesan.

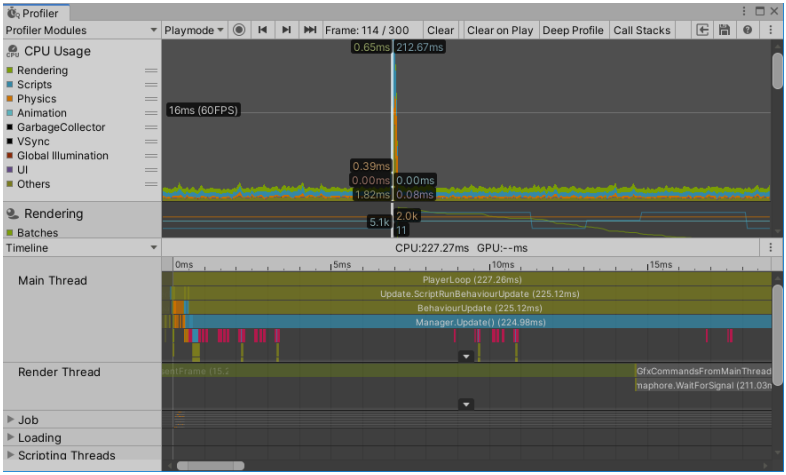
No.	*Average*	*Min*	*Max*
2000	21 (46.88ms)	0 (1047.36ms)	82 (12.15ms)
3000	21 (46.59ms)	0 (1014.92ms)	79 (12.63ms)
4000	21 (46.02ms)	0 (1095.05ms)	75 (13.16ms)
5000	20 (48.94ms)	1 (972.45ms)	82 (12.07ms)

6000	21 (46.18ms)	0 (1177.11ms)	86 (11.62ms)

Berdasarkan pengamatan pada data – data di atas, terlihat bahwa waktu pemrosesan terbaik diraih ketika menggunakan metode yang diajukan. Data – data di atas juga menunjukkan bahwa kolom *Average* tidak berkontribusi banyak dalam menunjukkan performa metode yang digunakan karena pemrosesan terjadi hanya pada beberapa *frame* saja di antara 1000 *frame* yang diambil. Kolom *Max* pun tidak memberikan data yang dapat diolah dikarenakan nilainya yang tidak stabil dan tidak sensitif seperti kolom *Min*. Jika kita amati lebih jauh, waktu yang dibutuhkan jika tidak menggunakan metode yang diajukan sama sekali memberikan hasil yang sama dengan tanpa menggunakan pemrosesan paralel. Hal ini dikarenakan pemrosesan paralel dan pemrosesan antar *frame* diisukan di *frame* yang berbeda di mana pemrosesan paralel dieksekusi terlebih dahulu dan akan mengantrekan hasilnya di *frame* yang sama. Namun pemrosesan antar *frame* hanya dieksekusi di *frame* berikutnya karena harus menunggu pemanggilan oleh *loop* utama.

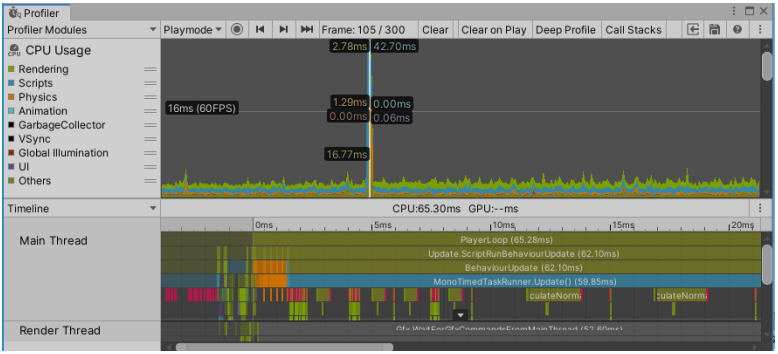


Gambar 5.1 Grafik yang dihasilkan menggunakan metode yang diajukan dan *perlin noise* pada perangkat *PC*.

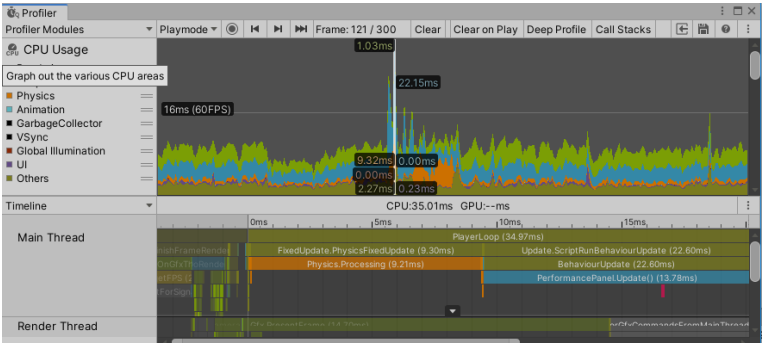


Gambar 5.2 Grafik yang dihasilkan tanpa menggunakan metode yang diajukan dan *perlin noise* pada perangkat *PC*.

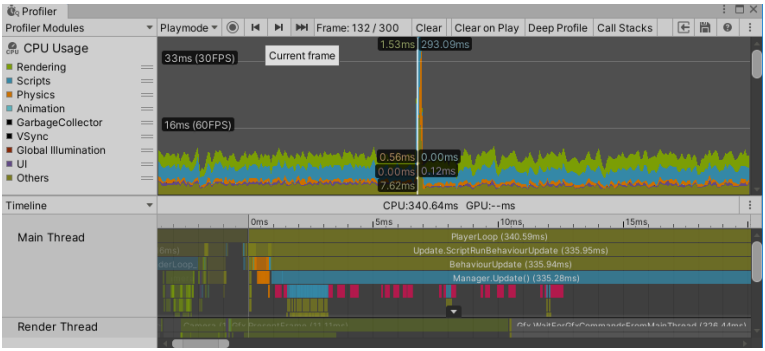
Gambar 5.3 Grafik yang dihasilkan menggunakan metode yang diajukan dan *simplex noise* pada perangkat *PC*.



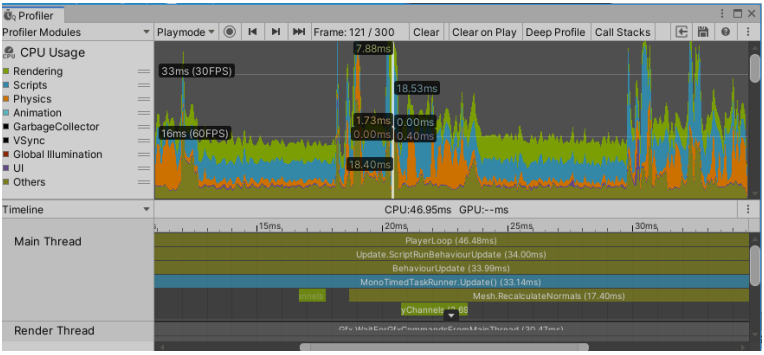
Gambar 5.4 Grafik yang dihasilkan tanpa menggunakan metode yang diajukan dan *simplex noise* pada perangkat *PC*.



Gambar 5.5 Grafik yang dihasilkan menggunakan metode yang diajukan pada perangkat *Xiaomi* 9T Pro.

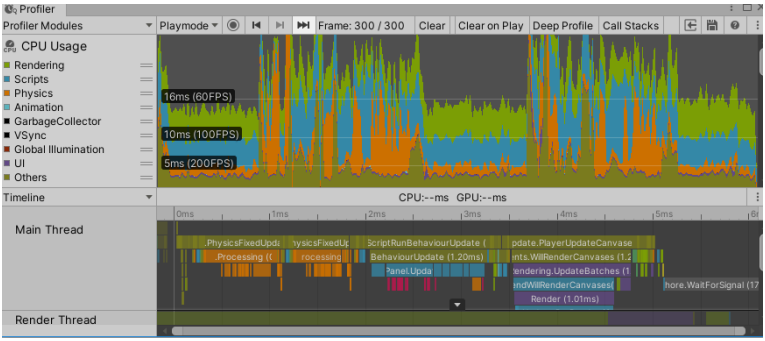


Gambar 5.6 Grafik yang dihasilkan tanpa menggunakan metode yang diajukan pada perangkat *Xiaomi* 9T Pro.

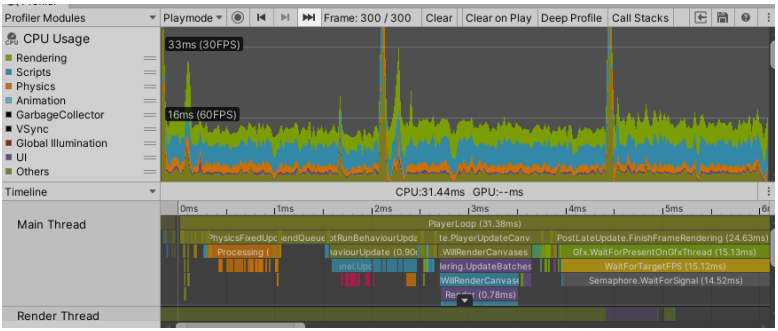


Gambar 5.7 Grafik yang dihasilkan menggunakan metode yang diajukan pada perangkat *Xiaomi* 4X.

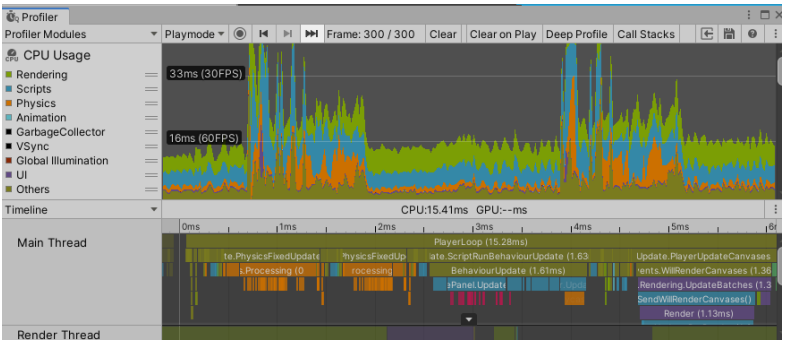
Gambar 5.8 Grafik yang dihasilkan menggunakan tanpa metode yang diajukan pada perangkat *Xiaomi* 4X.



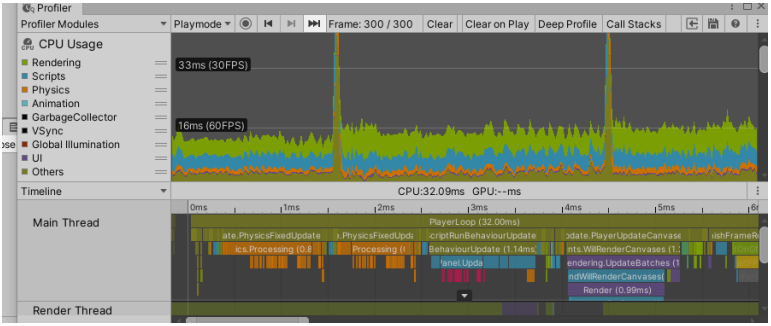
Gambar 5.9 Grafik yang dihasilkan menggunakan metode yang diajukan pada perangkat *Oppo* A57.



Gambar 5.10 Grafik yang dihasilkan menggunakan tanpa metode yang diajukan pada perangkat *Oppo* A57.



Gambar 5.11 Grafik yang dihasilkan menggunakan metode yang diajukan pada perangkat *Vivo* Y12.



Gambar 5.12 Grafik yang dihasilkan menggunakan tanpa metode yang diajukan pada perangkat *Vivo* Y12.

Berdasarkan data yang ditunjukkan oleh *profiler* dari *Unity*, kita dapat melihat kemajuan yang diberikan pada metode yang diajukan. Pada *profiler* kita hanya bisa melihat pemrosesan yang dilakukan di *thread* utama. Maka dari itu kita tidak dapat melihat pemrosesan yang dilakukan secara paralel. Namun berdasarkan data ini kita dapat mengetahui dampak dari penggunaan metode yang diajukan. Pada gambar 5.1, 5.5, 5.7, 5.9, dan 5.11 menunjukkan bahwa pemrosesan antar *frame* menyebabkan waktu eksekusi pada *frame* tersebut dan beberapa *frame* ke belakang menjadi lebih tinggi jika dibandingkan dengan *frame* lainnya. Gambar 5.5, 5.7, 5.9, dan 5.11 menunjukkan bahwa lebih banyak *frame* yang terpakai untuk memproses data tersebut dengan *frame* terbanyak ditunjukkan pada gambar 5.7 pada perangkat *Xiaomi* 4X. Terlihat bahwa jika tidak menggunakan metode pemrosesan yang diajukan terjadi pemrosesan yang membutuhkan waktu yang cukup lama namun terjadi pada beberapa *frame* saja seperti yang ditunjukkan pada gambar 5.2, 5.6, 5.8, 5.10 dan 5.12.

[Halaman ini sengaja dikosongkan]

BAB VI
KESIMPULAN

Pada bab ini akan dijelaskan kesimpulan dari hasil uji coba yang telah dilakukan.

Kesimpulan

Dari hasil uji coba yang telah dilakukan menggunakan metode yang diajukan didapati beberapa kesimpulan sebagai berikut:

1. Pemrosesan secara paralel memberikan dampak paling tinggi terhadap performa permainan
2. Pemrosesan antar *frame* memberikan dampak yang cukup tinggi terhadap performa permainan
3. Tidak semua pekerjaan dapat dilaksanakan secara paralel, dan meskipun perkerjaan tersebut sudah di pecah menjadi beberapa pekerjaan yang kecil masih ada kemungkinan pekerjaan tersebut memakan waktu yang cukup banyak
4. Meskipun sudah menggunakan metode yang diajukan, setiap perangkat tetap memiliki batasan kemampuan masing – masing
5. Penerapan pemrosesan paralel membutuhkan upaya yang lebih besar dari pada pemrosesan antar *frame*, hal ini dikarenakan pada pemrosesan paralel pendekatan yang dilakukan harus diubah menjadi pendekatan yang tidak sinkron.
6. Dunia yang dibuat merupakan dunia yang tidak memiliki batas pembuatan. Hal ini dikarenakan pembuatan petak selanjutnya tidak terikat dengan petak sebelumnya.

Saran

Saran yang diberikan terhadap pengembangan ke depannya adalah perlu diadakannya penyesuaian parameter untuk perangkat yang berbeda – beda. Penggunaan metode pemrosesan yang diajukan cocok digunakan pada aplikasi di mana respons yang

cepat sangat dibutuhkan namun hasil dari pemrosesan dapat ditunda seperti *virtual reality*.

DAFTAR PUSTAKA

[1] J. Bevins, “libnoise: What is coherent noise?,” [Online]. Available: http://priede.bf.lu.lv/ftp/pub/TIS/bibliotekas/libnoise/aprakstiHTML/coheren_noise.html. [Diakses 5 November 2019].

[2] D. M. D. Carli, F. Bevilacqua, C. T. Pozzer and M. C. d'Ornellas, A Survey of Procedural Content Generation Techniques Suitable to Game Development, Alegrete: IEEE, 2011.

[3] A. Doul, "What Pcg Is - Procedural Content Generation Wiki," [Online]. Available: pcg.wikidot.com/what-pcg-is. [Accessed 5 November 2019].

[4] N. Editor, "Unity Technologies - The World's Leading Game Engine," Nanalyze, [Online]. Available: https://www.nanalyze.com/2017/10/unity-technologies-leading-game-engine/. [Accessed 5 November 2019].

[5] O. S. Good, "It's impossible to visit every planet in No Man's Sky," Polygon, [Online]. Available: https://www.polygon.com/2014/8/19/6045933/its-impossible-to-visit-every-planet-in-no-mans-sky. [Accessed 5 November 2019].

[6] S. Gustavson, Simplex Noise Demystified, Linköping University, 2005.

[7] Intel, Intel Hyper-Threading Technology Technical User's Guide, Intel Corporation, 2003.

[8] Microsoft, "C# Guide Microsoft Docs," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/. [Accessed 5 November 2019].

[9] J. Peck, "Auburns_FastNoise Fast C++ Noise Library," [Online]. Available: https://github.com/Auburns/FastNoise. [Accessed 5 November 2019].

[10] I. Quilez, "Inigo Quilez fractals, computer graphics, mathematics, shaders, demoscene and more," [Online]. Available: https://www.iquilezles.org/www/articles/warp/warp.htm. [Accessed 5 November 2019].

[11] W. L. Raffe, F. Zambetta and X. Li, A Survey of Procedural Terrain Generation Techniques using Evolutionary Algorithms, Brisbane: IEEE, 2012.

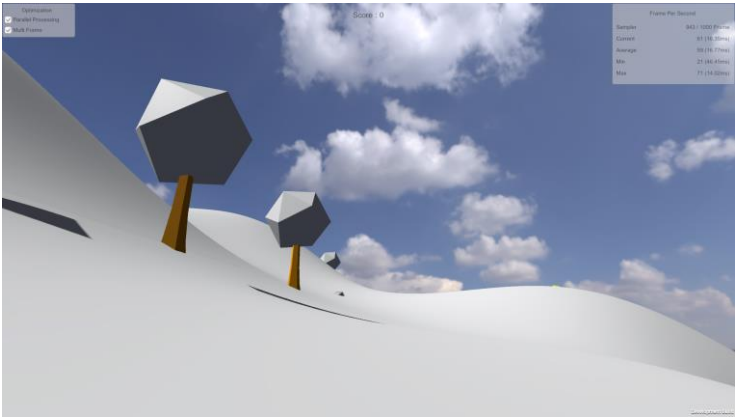
[12] R. J. Vitacon and L. Liu, Procedural Generation of 3D Planetary-Scale Terrains, Los Angeles: IEEE, 2019.

LAMPIRAN

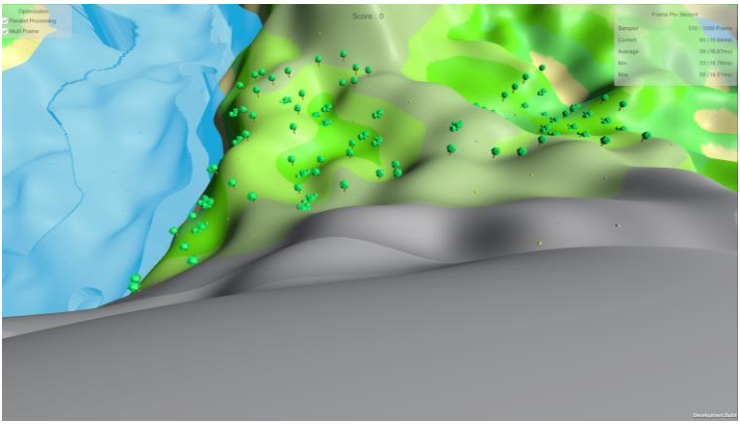
A. Tampilan Dalam Aplikasi

Berikut ini adalah tampilan dalam aplikasi :

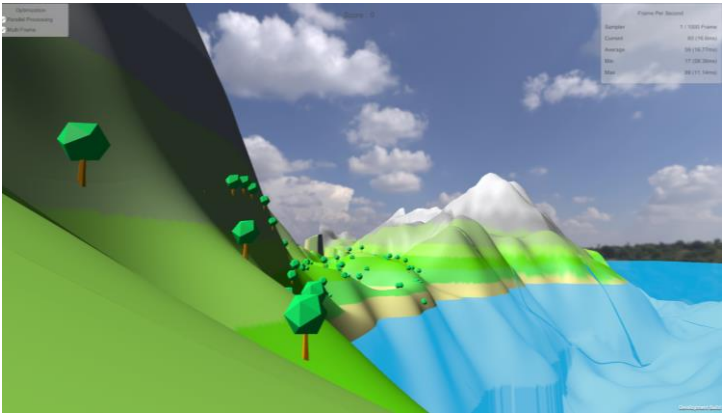
Gambar 5.13 Tampilan pada wilayah gunung.



Gambar 5.14 Tampilan pada wilayah gunung yang lain.



Gambar 5.15 Tampilan dari atas gunung.



Gambar 5.16 Tampilan pada tepi sungai.

BIODATA PENULIS

Firman Maulana, lahir pada tanggal 13 Juni 1998 di Surabaya. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Departemen Teknik Informatika Institut Teknologi Sepuluh Nopember (ITS). Memiliki beberapa hobi antara lain membuat perangkat lunak maupun menggabungkan perangkat keras sederhana. Penulis juga aktif berorganisasi di dalam kampus dengan menjadi staf pada himpunan. Di samping menjalankan pendidikan dan organisasi di kampus penulis juga mengerjakan proyek – proyek kecilnya sendiri. Email : maulana.firman56@gmail.com.