TUGAS AKHIR – EC184801

PROCEDURAL ENVIRONMENT GENERATION UNTUK MODEL 3D GUA MENGGUNAKAN OPENSIMPLEX NOISE DAN MARCHING CUBE

Khalil Satyadama
NRP 0721 16 4000 0019

Dosen Pembimbing
Dr. Supeno Mardi Susiki Nugroho S.T, M.T
Reza Fuad Rachmadi S.T, M.T, Ph.D

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

[Halaman ini sengaja dikosongkan].

TUGAS AKHIR – EC184801

PROCEDURAL ENVIRONMENT GENERATION UNTUK MODEL 3D GUA MENGGUNAKAN OPENSIMPLEX NOISE DAN MARCHING CUBE

Khalil Satyadama
NRP 0721 16 4000 0019

Dosen Pembimbing
Dr. Supeno Mardi Susiki Nugroho S.T, M.T
Reza Fuad Rachmadi S.T, M.T, Ph.D

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

[Halaman ini sengaja dikosongkan].

FINAL PROJECT – EC184801

PROCEDURAL ENVIRONMET GENERATION FOR CAVE 3D MODEL USING OPENSIMPLEX NOISE AND MARCHING CUBE

Khalil Satyadama
NRP 0721 16 4000 0019

Advisors
Dr. Supeno Mardi Susiki Nugroho S.T, M.T
Reza Fuad Rachmadi S.T, M.T, Ph.D

Department of Computer Engineering
Faculty of Intelligent Electrical and Informatics Technology
Sepuluh Nopember Institute of Technology
Surabaya 2020

[Halaman ini sengaja dikosongkan].

PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**Procedural Environment Generation untuk Model 3D Gua Menggunakan Open-Simplex Noise dan Marching Cube**" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, Juni 2020

Khalil Satyadama

NRP. 07211640000019

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

PROCEDURAL ENVIRONMENT GENERATION UNTUK MODEL 3D GUA MENGGUNAKAN OPENSIMPLEX NOISE DAN MARCHING CUBE

Tugas Akhir ini disusun untuk memenuhi salah satu syarat memperoleh gelar Sarjana Teknik di Institut Teknologi Sepuluh Nopember Surabaya

Oleh: Khalil Satyadama (NRP: 07211640000019)

Tanggal Ujian : 6 Juli 2020 Periode Wisuda : September 2020

Disetujui oleh:

Dr. Supeno Mardi Susiki Nugroho S.T, M.T (Pembimbing I)
NIP: 197003131995121001

Dr. Reza Fuad Rachmadi S.T, M.T (Pembimbing II)
NIP: 198504032012121001

Susi Juniastuti S.T, M.T (Penguji I)
NIP: 196506181999032001

Dr. Surya Sumpeno S.T, M.Sc (Penguji II)
NIP. 196906131997021003

Dr. Adhi Dharma Wibawa S.T, M.T (Penguji III)
NIP. 197605052008121003

Mengetahui
Kepala Departemen Teknik Komputer

Dr. Supeno Mardi Susiki Nugroho, ST., MT.
NIP. 197003131995121001

Halaman ini sengaja dikosongkan

ABSTRAK

Nama Mahasiswa	:	Khalil Satyadama
Judul Tugas Akhir	:	*Procedural Environment Generation* untuk Model 3D Gua Menggunakan *Open-Simplex Noise* dan *Marching Cube*
Pembimbing	:	1. Dr. Supeno Mardi Susiki Nugroho, S.T, M.T 2. Reza Fuad Rachmadi, S.T, M.T, Ph.D

Dalam pengembangan permainan video, lingkungan merupakan elemen terpenting dalam permainan video. Selain sebagai arena bermain selama permainan berlangsung, lingkungan juga berfungsi sebagai penambah estetika pada permainan video. Proses pembuatan lingkungan bermain di permainan video biasa dikerjakan oleh seorang *3D artist* dengan desain lingkungan berdasarkan *concept art* yang telah dibuat. Lama pembuatan model 3D lingkungan bergantung pada jumlah *(*3D artist) yang terlibat beserta kemampuan dari masing-masing *3D artist*, skala dan kompleksitas dari model 3D yang dibuat. Faktor tersebut memengaruhi banyaknya sumber daya yang harus digunakan untuk menyelesaikan model 3D tersebut, seperti *fee* untuk *artist* yang terlibat dan waktu pengerjaan.
Procedural environment generation (PEG) merupakan metode untuk menciptakan beragam lingkungan permainan dan salah satu cara untuk memangkas sumber daya dalam pengerjaan model 3D. Banyak riset mengenai PEG lingkungan gua yang telah dilakukan, namun kebanyakan dari riset yang telah dilakukan memiliki kekurangan, yaitu tidak ada campur tangan pengembang permainan untuk menyusun struktur lorong dari lingkungan gua. Pada tugas akhir ini diharapkan mampu menghasilkan PEG yang mampu memberi kendali penuh kepada pengembang terhadap struktur dari gua yang akan dibuat.

Kata Kunci : Procedural Environment Generation, Gua, Mesh Generation, Noise, Metode, Game Development

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji dan syukur kehadirat Tuhan Yang Maha Esa atas segala karunia-Nya, penulis dapat menyelesaikan penelitian ini dengan judul **Procedural Environment Generation untuk Model 3D Gua Menggunakan OpenSimplex Noise dan Marching Cube**.

Penelitian ini disusun dalam rangka pemenuhan bidang riset di Departemen Teknik Komputer ITS, Bidang Studi *Game Development*, serta digunakan sebagai persyaratan menyelesaikan pendidikan S1. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Keluarga, Ibu, Bapak dan Saudara tercinta yang telah memberikan dorongan spiritual dan material dalam penyelesaian buku penelitian ini.
2. Bapak Dr. Supeno Mardi Susiki Nugroho, S.T, M.T selaku Kepala Departemen Teknik Komputer, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember dan dosen pembimbing I yang selalu memberi arahan selama mengerjakan penelitian untuk tugas akhir ini.
3. Bapak Reza Fuad Rachmadi S.T, M.T, Ph.D selaku dosen pembimbing II yang selalu memberi arahan selama mengerjakan penelitian untuk tugas akhir ini.
4. Bapak-ibu dosen pengajar Departemen Teknik Komputer, atas pengajaran, bimbingan, serta perhatian yang diberikan kepada penulis selama ini.
5. Seluruh teman-teman dari Teknik Komputer, Laboratorium B401, dan B201 Teknik Komputer ITS.

Kesempurnaan hanya milik Allah SWT, untuk itu penulis memohon segenap kritik dan saran yang membangun. Semoga penelitian ini dapat memberikan manfaat bagi kita semua. Amin.

Surabaya, April 2020

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

Abstrak

KATA PENGANTAR

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

NOMENKLATUR

1 PENDAHULUAN **1**
1.1 Latar belakang 1
1.2 Permasalahan 3
1.3 Tujuan 3
1.4 Batasan masalah 3
1.5 Sistematika Penulisan 4
1.6 Relevansi 5

2 TINJAUAN PUSTAKA **7**
2.1 Morfologi Gua 7
2.2 Garis Tangent Eksternal Dua Lingkaran 7
2.3 Simplex Noise 8
2.4 OpenSimplex Noise 10
2.5 Marching Cube 10
2.6 Compute Shader 11

3 DESAIN DAN IMPLEMENTASI SISTEM **13**
3.1 Cakupan Tugas Akhir 13
3.2 Desain Sistem 14
3.3 Alur Kerja 14
3.3.1 Pengaturan Posisi Titik Sampel dan *Chunk* . 15
3.3.2 Perancangan Garis Jalur Gua 17
3.3.3 Perancangan Input Noise 20

3.3.4 Perhitungan Titik Sampel yang Terjangkau Garis 21
3.3.5 Perhitungan Nilai *Noise* untuk Mendapatkan Sampel Volume 27
3.3.6 Rekonstruksi Model 3D 32

4 PENGUJIAN DAN ANALISIS 39
4.1 Perubahan Resolusi Voxel 41
4.2 Perubahan Parameter Noise 42
4.2.1 *Octave* 43
4.2.2 *Persistence* 44
4.2.3 *Lacunarity* 44
4.3 Perubahan Jumlah Jalur Gua 47
4.4 Rekonstruksi Kelima Jenis Gua 48
4.5 Survei Kepuasan Pengguna 49

5 PENUTUP 53
5.1 Kesimpulan 53
5.2 Saran 54

DAFTAR PUSTAKA 55

Biografi Penulis 57

DAFTAR GAMBAR

2.1 Keempat jenis jalur gua; *branchwork* (A), *network* (B), *anastomotic* (C), *ramiform* dan *spongework* (D). 8
2.2 Sebuah vekor tangent eksternal terhadap kedua lingkaran dengan radius dan posisi berbeda. 9
2.3 Hasil transformasi geser untuk pemetaan ke koordinat simplex. 9
2.4 Operasi *dot product* untuk mendapatkan nilai *noise*. 10
2.5 Perbedaan hasil transformasi geser pada *Simplex noise* dan *OpenSimplex noise*. 11

3.1 *Flowchart* proses sintesis model 3D. 15
3.2 Sebuah *chunk* dengan jumlah voxel sebanyak 800 unit. 17
3.3 Perbedaan ukuran voxel memengaruhi ukuran *chunk*. 17
3.4 *Chunk* yang sudah dibuat. 17
3.5 Ruang dari setiap jalur gua yang ditampilkan pada bidang biru. 18
3.6 Rentang yang dimaksud pada syarat pertama. 22
3.7 Rincian vektor yang menyusun frustum kerucut. . . 22
3.8 Tiga kemungkinan kondisi titik sampel berdasarkan lokasinya terhadap sebuah jalur gua. 23
3.9 Proyeksi titik sampel *Sp* ke objek *Line*. 24
3.10 Vektor T sejajar dengan vektor tangent terhadap dua lingkaran. 25
3.11 Representasi *noise* pada koordinat polar. 28
3.12 Representasi volume ruang gua dari gambar 3.11 sebelum dilakukan perhitungan *falloff*. 29
3.13 Daerah jalan yang ditunjukkan pada bagian berwarna biru. 29
3.14 Permukaan gua tanpa daerah jalan dan dengan daerah jalan. 30
3.15 Contoh sampel volume ruang gua. 31
3.16 Pemberian indeks pada vertex voxel dan titik tengah rusuk voxel. 33
3.17 *Sub-mesh* yang dihasilkan pada kombinasi 40. 34

3.18 Proses *marching cube* setelah membandingkan sampel volume dengan nilai *iso level*. 34
3.19 Perbandingan model 3D tanpa proses interpolasi dengan model 3D yang sudah diinterpolasi [1]. 35
3.20 Keempat vertex yang tertumpuk pada dua posisi berbeda. 36

4.1 Konfigurasi kamera dan pencahayaan pada *Blender*. 41
4.2 Perbandingan sebagian bagian model 3D menurut perubahan resolusi voxel. 42
4.3 Perbandingan sebagian bagian model 3D menurut perubahan nilai *octave*. 43
4.4 Perbandingan sebagian bagian model 3D menurut perubahan nilai *persistence*. 45
4.5 Pengaruh perubahan *persistence* pada simulasi oleh *Desmos*. 45
4.6 Perbandingan sebagian bagian model 3D menurut perubahan nilai *lacunarity*. 46
4.7 Pengaruh perubahan *lacunarity* pada simulasi oleh *Desmos*. 46
4.8 Konfigurasi jalur-jalur gua dari sudut pandang atas. A: *Branchwork*, B: *Network*, C: Anastomotik, D: *Ramiform*, E: *Spongework*. 48
4.9 Hasil rekonstruksi kelima jenis gua dari sudut pandang atas. A: *Branchwork*, B: *Network*, C: Anastomotik, D: *Ramiform*, E: *Spongework*. 49
4.10 Daerah dengan garis merah adalah artifak melayang yang tidak diinginkan. 50

DAFTAR TABEL

4.1 Tabel atribut objek *Node* 40
4.2 Tabel atribut objek *Line* 40
4.3 Tabel pengujian perubahan resolusi voxel. 41
4.4 Tabel pengujian perubahan parameter *octave* untuk input *noise*. 43
4.5 Tabel pengujian perubahan parameter *persistence* untuk input *noise*. 44
4.6 Tabel pengujian perubahan parameter *lacunarity* untuk input *noise*. 46
4.7 Tabel hasil pengujian perubahan jumlah jalur gua. . 47
4.8 Daftar pernyataan pada kuesioner. 51
4.9 Hasil survei pengguna. 52

Halaman ini sengaja dikosongkan

NOMENKLATUR

$noiseVal$	: Nilai *noise* yang dihasilkan.
$octave$	: Nilai yang menyatakan jumlah perulangan perhitungan *noise*.
$offset$	: Offset posisi untuk input*noise*.
$octOff$	: Offset posisi untuk input di tiap *octave*.
L	: Nilai *lacunarity noise*.
P	: Nilai *persistence noise*.
R_A	: Ujung garis gua dengan radius terkecil.
R_B	: Ujung garis gua dengan radius terbesar.
θ	: Sudut antara selimut frustum kerucut dengan garis jalur gua.
l	: Panjang vektor sebuah jalur gua.
p	: Panjang proyeksi orthogonal titik sampel ke jalur gua.
lr	: Interpolasi radius dari kedua ujung frustum kerucut berdasarkan proyeksi orthogonal titik sampel.
$nDist$	: Nilai lr yang digabungkan dengan *noise*.
$simulatePathway$	: Variabel yang menentukan apakah simulasi untuk membuat daerah jalan dilakukan atau tidak.
$volume$	: Sampel volume yang diperoleh titik sampel.
$isoLevel$	: *Threshold* data volume gua.
i	: Interpolan untuk interpolasi titik tengah rusuk voxel.
pos	: Posisi titik tengah rusuk voxel.

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

Penelitian ini di latar belakangi oleh berbagai kondisi yang menjadi acuan. Selain itu juga terdapat beberapa permasalahan yang akan dijawab sebagai luaran dari penelitian.

1.1 Latar belakang

Proses pengembangan *video game* tak pernah lepas dari keterlibatan seluruh anggota tim pengembang *video game*. Setiap anggota juga memiliki perannnya masing-masing, misalnya seorang *level designer* mendesain sebuah *level* sedemikian rupa agar setiap *level* memiliki karakteristik tersendiri dan tingkat kesulitan yang beragam sehingga mampu meningkatkan tingkat keseruan bermain bagi pemain. Tak kala kualitas dari hasil pekerjaan juga bergantung pada *skill* dari masing-masing anggota, kemampuan komunikasi, dan biaya yang diperlukan dalam proses produksi [2].

Salah satu faktor yang mampu meningkatkan tingkat kepuasan bermain oleh pemain adalah faktor estetika pada permainan [3] [4]. Estetika sebuah permainan sangat nampak pada objek yang berinteraksi dengan pemain dan juga lingkungan yang menjadi area bermain. Estetika juga tak lepas dari unsur tema yang dibawa oleh tim pengembang, contohnya seri permainan *Silent Hill* yang membawa tema horror, surealisme, dan mitologi diiringi dengan visual lingkungan yang gelap, sempit, dan berkesan tidak nyaman dilihat bagi sebagian besar orang. Contoh lainnya adalah seri permainan *Kirby* yang membawa tema fantasi, petualangan, dan keceriaan sehingga desain visual yang ditampilkan haruslah berwarna, lucu, dan menarik bagi semua kalangan.

Masalah estetika ini akan membesar ketika skala dari permainan yang dibuat semakin besar. Permainan untuk konsol lama seperti *Nintendo Entertainment System*, dikarenakan oleh terbatasnya memori pada konsol tersebut, para pengembang harus menemukan solusi untuk mampu menciptakan *map* sebuah *level* sedemikian rupa tanpa menghabiskan banyak memori untuk *sprite* lingkungan. Selain masalah pada perangkat keras, tim pengembang juga perlu

mengatur sumber daya yang mereka miliki untuk membuat konten pada permainan mereka, termasuk anggota tim yang mengurus masalah estetika dan aset permainan. Tak jarang juga pengembang permainan membutuhkan *freelance artist* untuk membantu mengerjakan aset permainan, seperti *freelance artist* dengan nama pena *alchemaniac* sempat menjadi desainer karakter untuk permainan *Valiant Force* (*XII Braves*, 2016-2019) dan *Arknights* (*Yostar*, 2019-sekarang).

Salah satu solusi untuk masalah tersebut adalah dengan menerapkan metode *procedural content generation* (PCG), yaitu sebuah metode untuk membuat konten berdasarkan sebuah algoritma. Konten yang dihasilkan bisa diterapkan langsung pada permainan tersebut seperti *Minecraft* dan *Terraria* atau diolah kembali oleh tim pengembang untuk menambahkan konten lain sesuai desain tim. Banyak manfaat yang dapat diperoleh dari pengaplikasian *procedural content generation* pada pengembangan permainan, seperti pemangkasan biaya dan waktu produksi, cepatnya proses pembuatan konten seiring berkembangnya teknologi komputasi, dan kemudahan membuat konten yang beragam hanya dengan mengubah nilai dari beberapa parameter.

Terdapat beberapa jenis PCG, salah satunya adalah *procedural environment generation* (PEG). Sesuai dengan namanya, jenis PCG ini digunakan untuk menghasilkan sebuah aset lingkungan berdasarkan algoritma yang telah dirancang. Tentunya, PEG memiliki beberapa jenis sesuai dengan lingkungan apa yang akan dibuat. Sementara itu, penelitian mengenai PEG telah banyak dilakukan untuk menemukan berbagai pendekatan dan mencari kelebihan serta kekurangan dari metode yang ditawarkan. Pengaplikasian dan penelitian PEG untuk membuat daerah daratan di atas permukaan Bumi telah banyak dilakukan, namun jarang ditemukan baik penelitian maupun pengaplikasian PEG untuk membuat daerah di bawah permukaan Bumi, seperti gua. Telah dilakukan setidaknya dua penelitian yang membahas PEG untuk membuat aset lingkungan gua; penelitian pertama melakukan pendekatan dengan membuat lingkungan gua dalam sebuah bangun ruang bola yang kemudian diberikan nilai *noise* untuk mengubah bentuk dari permukaan gua yang dibuat [5] [6], penelitian kedua melakukan pendekatan dengan

membuat lingkungan gua dengan menyiapkan data berupa jalur-jalur gua yang dibuat otomatis menggunakan algoritma *L-system*. Setiap jalur gua berisi beberapa *metaball* dengan ukuran tertentu yang permukaannya sudah terdistorsi dengan *simplex noise* dan *Voronoi noise* [7].

1.2 Permasalahan

Dari kedua penelitian yang telah disebutkan, masing-masing memiliki kekurangan. Ruang gua yang dihasilkan pada penelitian pertama masih terbatas pada bangun ruang bola, sedangkan pada penelitian kedua tidak ditampilkan *preview* dari permukaan gua yang akan dihasilkan sehingga para pengguna tidak mengetahui bagaimana bentuk permukaan gua yang dihasilkan. Permasalahan lainnya adalah tidak adanya akses bagi pengguna untuk mengubah jalur-jalur gua dari hasil algoritma *L-system*. Hal ini menghilangkan kebebasan pengguna untuk membuat sebuah gua berdasarkan desain gua yang telah dibuat dengan mempertimbangkan beberapa aspek seperti tingkat kesulitan bermain berdasarkan *map* sebuah *stage/ level*.

1.3 Tujuan

Dari perumusan masalah tersebut, penelitian ini bertujuan untuk mengembangkan metode *procedural content generation environment* gua yang meminimalkan kinerja pengembang, khususnya para *artist*, dalam menentukan bentuk dari model 3D gua yang akan dibuat dengan mengubah sejumlah parameter untuk membuat data volume gua, serta mengatasi kekurangan dan mengambil kelebihan dari kedua penelitian, yaitu bentuk ruang gua yang tidak hanya terbatas pada ruang bola, namun berdasarkan ruang dari jalur-jalur gua yang telah disediakan dan menghasilkan sebuah ruang gua yang tidak memiliki artifak melayang di tengah-tengah ruang gua.

1.4 Batasan masalah

Batasan masalah yang timbul dari permasalahan Tugas Akhir ini adalah:

1. Berfokus pada proses pembuatan model 3D dari *environment* gua, tidak sampai pada proses *UV-mapping* dan pembuatan

material model 3D

2. Penelitian dilakukan dengan menggunakan aplikasi *game engine* Unity versi 2019.3.0f3.
3. Fitur-fitur gua (speleothem) seperti stalagtit, stalagmit, dan deposit sekunder lainnya tidak disertakan.

1.5 Sistematika Penulisan

Laporan penelitian Tugas akhir ini tersusun dalam sistematika dan terstruktur sehingga mudah dipahami dan dipelajari oleh pembaca maupun seseorang yang ingin melanjutkan penelitian ini. Alur sistematika penulisan laporan penelitian ini yaitu:

1. BAB I Pendahuluan
   Bab ini berisi uraian tentang latar belakang permasalahan, penegasan dan alasan pemilihan judul, sistematika laporan, tujuan dan metodologi penelitian.
2. BAB II Dasar Teori
   Bab ini menjelaskan teori dari sebuah gua untuk menentukan jenis gua seperti apa yang akan kami buat, teori garis tangent eksternal terhadap dua lingkaran yang akan membantu tahap perhitungan sampel volume, penjelasan algoritma *Simplex noise*, dan penjelasan *compute shader* untuk pengolahan sampel volume dan konversi sampel volume menjadi model 3D.
3. BAB III Perancangan Sistem dan Impementasi
   Bab ini berisi desain sistem kami beserta implementasinya. Setiap implementasi tahap disertakan penjelasan secara matematis dan penerapannya pada program yang dibuat melalui *pseudocode*.
4. BAB IV Pengujian dan Analisis
   Bab ini menjelaskan tentang pengujian metode yang diimplementasikan terhadap hasil dari model 3D yang dibuat dan lama waktu yang dibutuhkan untuk menghasilkan model 3D, serta survei kepuasan pengguna terhadap program yang telah kami buat.
5. BAB V Penutup
   Bab ini merupakan penutup yang berisi kesimpulan yang di-

ambil dari penelitian dan pengujian yang telah dilakukan. Saran dan kritik yang membangun untuk pengembangkan lebih lanjut juga dituliskan pada bab ini.

1.6 Relevansi

Penelitian mengenai *procedural content generation* merupakan bidang penelitian yang sangat diperlukan untuk memudahkan para *game developer* dalam membuat konten yang akan mereka masukkan ke dalam *game* mereka. Di luar bidang *game development*, *procedural content generation* dapat digunakan untuk keperluan lain seperti pembuatan aset pada animasi 3D dan *background* untuk *digital painting*.

Halaman ini sengaja dikosongkan

BAB 2
TINJAUAN PUSTAKA

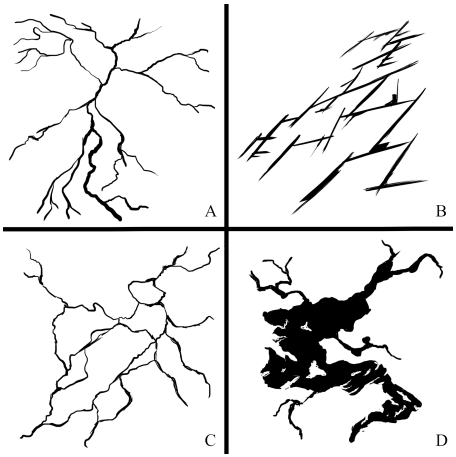
Demi mendukung penelitian ini, dibutuhkan beberapa teori penunjang sebagai bahan acuan dan referensi. Dengan demikian penelitian ini menjadi lebih terarah.

2.1 Morfologi Gua

Untuk mengetahui jenis gua apa yang akan kami buat, kami perlu mengetahui morfologi sebuah gua. Sebuah gua terdiri dari sekumpulan lorong-lorong yang bersinggungan satu sama lain sehingga membentuk sebuah pola. Pola ini dapat diketahui berdasarkan ekspedisi yang dilakukan di dalam gua. Setidaknya ada lima macam pola gua yang diketahui, yaitu *branchwork*, *network*, *anastomotic*, *ramiform*, dan *spongework*. *Branchwork* adalah jenis gua yang sering ditemukan pada gua-gua di dunia nyata, di mana setiap jalur gua dapat berhubungan dengan jalur lain membentuk sebuah sungai-sungai kecil (*tributary*). Pada jenis gua ini jarang ditemukan kumpulan jalur gua yang membentuk sebuah *loop*. Kebalikannya dapat ditemui pada gua jenis *anastomotic*. Pada gua *network* terdiri dari jalur-jalur yang membentuk struktur berpetak-petak atau mirip struktur yang ditemukan pada peta daerah perkotaan. Ketiga jenis gua yang telah disebutkan adalah jenis gua dengan bentuk ruang gua yang relatif sama pada setiap jalurnya. Sebaliknya, jenis *ramiform* dan *spongework* memiliki bentuk ruang gua yang bervariasi pada setiap jalurnya [8].

2.2 Garis Tangent Eksternal Dua Lingkaran

Teori ini mendeskripsikan garis-garis yang bersinggungan dengan sebuah lingkaran tepat pada satu titik dan tidak memasuki daerah dalam lingkaran yang disebut garis tangent. Garis tangent terhadap dua lingkaran dapat dibagi menjadi dua jenis, yaitu internal dan eksternal. Internal menyinggung dua lingkaran dan melewati vektor yang menghubungkan antar titik pusat dari kedua lingkaran, sedangkan eksternal hanya menyinggung kedua lingkaran saja. Pada dua dimensi, keempat vektor yang terdiri dari garis tangent eksternal, garis antara dua titik pusat lingkaran, dan



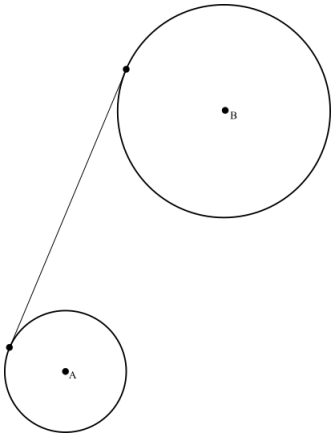
Gambar 2.1: Keempat jenis jalur gua; *branchwork* (A), *network* (B), *anastomotic* (C), *ramiform* dan *spongework* (D).

dua garis yang menghubungkan titik singgung garis tangent dengan lingkaran dan titik pusat lingkaran akan membentuk sebuah bidang trapesium yang akan memudahkan perhitungan seperti vektor antara dua titik singgung dari dua garis tangent, sudut antara garis titik singgung dan titik pusat lengkaran dengan garis dari dua titik pusat lingkaran, beserta jarak titik pusat dengan garis antara dua titik singgung [9]. Bidang trapesium ini juga akan mendefinisikan jangkauan dari satu buah garis jalur [10].

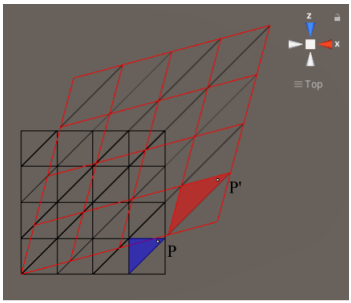
2.3 Simplex Noise

Simplex noise merupakan pengembangan dari algoritma *Perlin noise*. Input yang diberikan berupa koordinat titik dalam dimensi dua atau dimensi tiga. Transformasi geser (*skew*) dijalankan untuk memetakan titik koordinat tersebut kedalam koordinat simplex. Faktor geser untuk pemetaan pada koordinat simplex ditentukan pada rumus berikut:

$$F = \frac{\sqrt{n+1} - 1}{n} \tag{2.1}$$

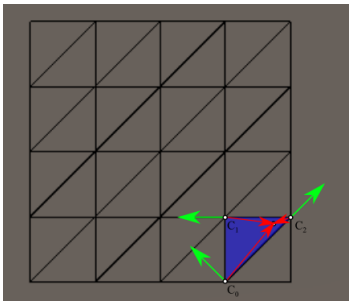


Gambar 2.2: Sebuah vekor tangent eksternal terhadap kedua lingkaran dengan radius dan posisi berbeda.



Gambar 2.3: Hasil transformasi geser untuk pemetaan ke koordinat simplex.

Satu petak simplex terdiri dari n+1 titik pojok di mana n adalah dimensi dari koordinat input dan setiap titik pojok memiliki satu nilai vektor acak yang disebut sebagai vektor gradien. Nilai koordinat hasil transformasi geser akan dibulatkan untuk mengetahui titik koordinat berada pada petak simplex index ke berapa. Posisi dari semua titik pojok pada petak simplex yang sudah diketahui dalam koordinat simplex akan digeser untuk dipetakan ke koordinat di mana titik input berada. Faktor geser untuk pemetaan pada



Gambar 2.4: Operasi *dot product* untuk mendapatkan nilai *noise*.

koordinat semula ditentukan pada rumus berikut:

$$F' = \frac{1 - \frac{1}{\sqrt{n+1}}}{n} \tag{2.2}$$

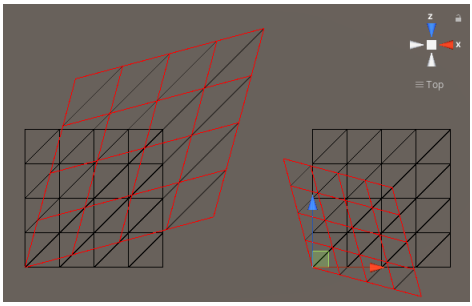
Terakhir, perhitungan *dot product* antara vektor titik input ke setiap titik pojok simplex dengan vektor gradien setiap titik pojok simplex dilakukan untuk mendapatkan output dari *simplex noise* [11][12].

2.4 OpenSimplex Noise

Dipublikasikan pada tahun 2014 oleh Kurt Spencer, *OpenSimplex noise* merupakan alternatif dari *Simplex noise* untuk mengatasi masalah penggunaan paten algoritma *Simplex noise*. *OpenSimplex noise* menggunakan konstanta yang berbeda untuk transformasi geser dan jumlah titik pojok simplex yang lebih banyak dibandingkan *Simplex noise* sehingga didapatkan *noise* yang lebih halus.

2.5 Marching Cube

Marching cube merupakan algoritma untuk rekonstruksi data volume menjadi model 3D oleh William E. Lorensen dan Harvey E. Cline [13]. Sebuah voxel didefinisikan dengan delapan titik yang memiliki informasi posisi dan aktif tidaknya titik tersebut yang membentuk bangun ruang kubus. Dari kombinasi nilai aktif tidaknya dari kedelapan titik tersebut, terdapat 255 kombinasi *sub-mesh*



Gambar 2.5: Perbedaan hasil transformasi geser pada *Simplex noise* dan *OpenSimplex noise*.

yang terbentuk. Kumpulan dari *sub-mesh* yang dihasilkan akan membentuk sebuah model 3D utuh. Operasi *welding* dapat dilakukan secara opsional untuk menyatukan dua atau lebih vertex yang memiliki nilai posisi yang berdekatan [14].

2.6 Compute Shader

Compute shader adalah *shader* yang dapat diprogram untuk melakukan komputasi di luar komputasi *rendering*, seperti *rasterization* oleh *fragment shader* [15]. Sebuah modifikasi dari *compute shader* oleh Unity menambahkan parameter *#pragma kernel* untuk menyatakan fungsi mana yang akan dieksekusi ketika *compute shader* dijalankan (*dispatch* [16]. Saat *compute shader* dijalankan, sejumlah *dispatch thread* dibuat di mana setiap *dispatch thread* terdiri dari kumpulan *thread group* [17].

Halaman ini sengaja dikosongkan

BAB 3
DESAIN DAN IMPLEMENTASI SISTEM

Penelitian ini dilaksanakan sesuai dengan desain sistem berikut dengan implementasinya. Desain sistem merupakan konsep dari pembuatan dan perancangan infrastruktur kemudian diwujudkan dalam bentuk blok-blok alur yang harus dikerjakan. Pada bagian implementasi merupakan pelaksanaan teknis untuk setiap blok pada desain sistem.

3.1 Cakupan Tugas Akhir

Penelitian ini dilakukan untuk mengembangkan metode *procedural environment generation* untuk lingkungan gua. Telah dilakukan penelitian-penelitian mengenai *procedural environment generation* lingkungan gua yang memiliki kelebihan dan kekurangan. Penelitian yang dilakukan oleh Juncheng Cui menerapkan metode *octree* untuk mengonversi sampel volume ruang gua menjadi model 3D [5][6]. Dengan menggunakan metode *octree*, masalah artifak-artifak melayang di tengah-tengah ruang gua akan teratasi. Namun begitu, model 3D yang dihasilkan dari kedua penelitian tersebut masih dibuat dalam sebuah ruang menyerupai bola. Sementara itu, model 3D gua yang dihasilkan menggunakan pendekatan oleh Benjamin Mark sudah mendekati ruang gua yang ada di dunia nyata [7]. Pendekatan ini menggunakan algoritma *L-system* untuk menghasilkan jalur-jalur gua yang bervariasi, kemudian pada salah satu ujung sebuah jalur terdapat *metaball*, dengan ukuran yang telah ditentukan, yang terdistorsi oleh sebuah *noise*. *Metaball* ini akan bergerak menuju ujung jalur gua lainnya yang kemudian akan membentuk sebuah volume dari ruang gua. Konversi dari volume gua menjadi model 3D menggunakan algoritma *marching cube*. Dari metode ini masih terdapat kekurangan yaitu jalur gua yang dibuat tidak bisa diubah-ubah secara manual apabila pada situasi tertentu saat proses pembentukan lingkungan gua membutuhkan perubahan jalur gua.

Dari kelebihan dan kekurangan penelitian yang telah disebutk-

an, hasil akhir dari penelitian ini adalah menghasilkan sebuah *tools* yang mampu menghasilkan model 3D gua tanpa masalah artifak-artifak melayang dalam ruang gua dan memberi kemudahan bagi pengguna *tools* untuk mengubah konfiguasi dari jalur-jalur gua. Adapun batasan masalah dari penelitian ini, yaitu:

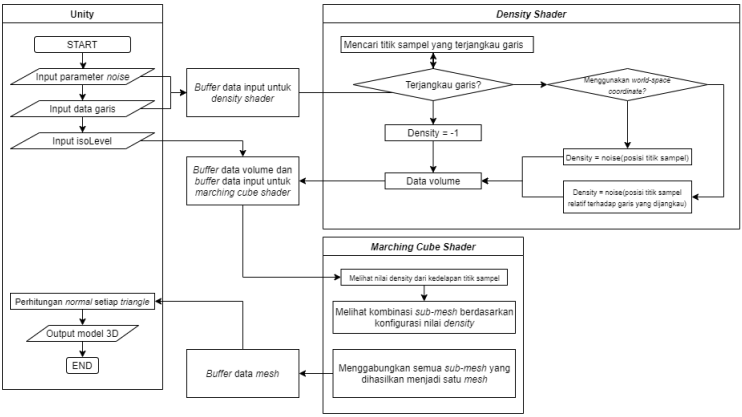
1. Penelitian ini tidak membahas proses pemberian *material* atau *texture* secara otomatis pada model 3D gua. Model 3D gua pada dunia nyata memiliki ragam warna permukaan yang berbeda yang didapat dari jenis material yang menyusun permukaan gua.
2. Penelitian ini tidak membahas bagaimana speleothem gua terbentuk. Penelitian ini berfokus untuk menghasilkan permukaan gua sederhana dari metode yang kami ajukan. Speleothem adalah fitur-fitur yang terdapat pada permukaan gua. Speleothem yang sering terdapat pada sebuah gua adalah stalaktit dan stalakmit.

3.2 Desain Sistem

Pada tugas akhir ini, Unity digunakan sebagai media untuk input, visualisasi, dan pengolahan model 3D sekaligus. *Compute shader* yang digunakan akan dibagi menjadi dua jenis berdasarkan outputnya, yaitu *density shader* dan *marching cube shader*. Proses kerja dari sistem ini akan dijelaskan melalui *flowchart* pada gambar 3.1. Proses pertama yang dilakukan oleh Unity, setelah mendapatkan input, adalah meletakkan data input ke dalam sebuah *buffer* agar *density* dan *marching cube shader* dapat membaca input yang diberikan pada Unity. Pada *density shader*, akan dicari titik sampel yang terjangkau oleh seluruh garis jalur gua. Apabila titik sampel tidak terjangkau oleh satu garis, nilai volume pada titik sampel secara otomatis diberi nilai -1. Sebaliknya, nilai volume titik sampel akan dihitung berdasarkan nilai *noise* yang didapatkan berdasarkan posisi garis. Setelah mendapatkan data volume, langkah selanjutnya adalah mengonversi data volume menjadi 3D dengan menggunakan metode *marching cube*.

3.3 Alur Kerja

Pembuatan tugas akhir ini dibagi menjadi beberapa tahapan, yaitu:



Gambar 3.1: *Flowchart* proses sintesis model 3D.

1. Pengaturan Posisi Titik Sampel dan *Chunk*.
2. Perancangan Jalur Gua.
3. Perancangan Input *Noise*.
4. Perhitungan Titik Sampel yang Terjangkau Garis.
5. Perhitungan Nilai *Noise* untuk Mendapatkan Sampel Volume.
6. Rekonstruksi Model 3D.

3.3.1 Pengaturan Posisi Titik Sampel dan *Chunk*

Langkah ini dilakukan untuk mengatur lokasi titik sampel saat digunakan pada tahap selanjutnya. Mengingat keterbatasan *Unity* terkait jumlah vertex model 3D pada tiap objek yang dapat diproses, proses ini diperlukan untuk membatasi jumlah titik sampel, yang kemudian membatasi jumlah voxel untuk proses *marching cube*, sehingga dari model 3D yang dihasilkan tidak melampaui keterbatasan tersebut. Model 3D gua yang akan dibuat ditampung dalam kumpulan objek yang disebut sebagai *chunk*, di mana setiap objek *chunk* menampung beberapa voxel. Baik voxel maupun *chunk* berbentuk bangun ruang kubus. Panjang sisi dari objek *chunk* didefinisikan sebagai jumlah unit voxel yang terdapat pada tiap sumbu di koordinat tiga dimensi. Adapun jumlah vertex di setiap sumbu

XYZ objek *chunk* senilai:

$$\Sigma_{vertex_{XYZ}} = \Sigma_{voxel_{XYZ}} + 1 \tag{3.1}$$

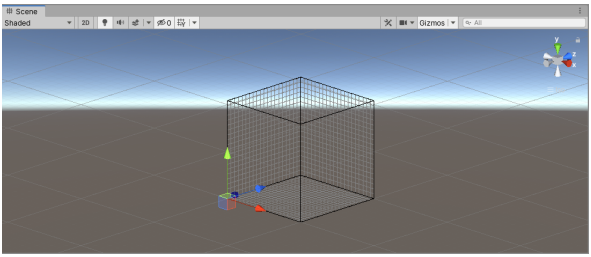
Dikarenakan keterbatasan pada *game engine* Unity, jumlah elemen *vertex buffer* model 3D yang bisa ditampung hanya sebanyak 65535 vertex, sehingga jumlah voxel yang bisa ditampung oleh objek *chunk* adalah:

$$\Sigma_{voxel_{max}} = \frac{65535}{5} = 13107 \tag{3.2}$$

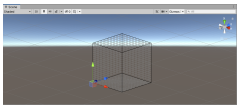
Nilai 5 didapat dari jumlah vertex terbanyak dari kombinasi *submesh* pada algoritma *marching cube*. Karena bentuk *chunk* adalah kubus, maka *chunk* hanya mampu menampung 23 voxel di sumbu XYZ. Atas keterbatasan ini, apabila akan membuat model 3D gua yang lebih besar dan lebih detil, *chunk* yang harus dibuat semakin banyak.

Jumlah *chunk*, sebagaimana jumlah voxel pada sebuah *chunk* didefinisikan, diatur dengan mengubah parameter resolusi yang mendefinisikan jumlah *chunk* pada tiap sumbu tiga dimensi. Ukuran sebuah *chunk* bergantung pada jumlah voxel dan ukuran unit voxel yang ditampung. Ukuran unit voxel juga mendefinisikan jarak kedua vertex voxel. Baik vertex voxel, voxel, maupun *chunk* akan diberi indeks untuk memudahkan pengaksesan properti seperti sampel volume di setiap vertex voxel pada sebuah *compute buffer*. Dengan perancangan ini, titik sampel telah teratur dalam vertex voxel dan terbentuk susunan petak seperti gambar 3.2. Gambar 3.5 menunjukkan *chunk* yang dibuat dengan resolusi 8x5x6 pada sumbu XYZ dan ukuran voxel adalah 0.5 unit *Unity*. Properti-properti *chunk* akan disimpan dalam sebuah file *compute shader* berikut.

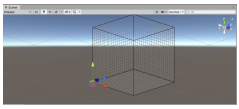
```
//file: Inputs.compute
RWStructuredBuffer<float4> samplePoints;  //buffer dengan
     akses read dan write yang berisi seluruh vertex
    voxel/ titik sampel
int1 numVertexX, numVertexY, numVertexZ;  //jumlah vertex
     pada setiap chunk
float1 voxelSize;       //ukuran voxel unit
```



Gambar 3.2: Sebuah *chunk* dengan jumlah voxel sebanyak 800 unit.

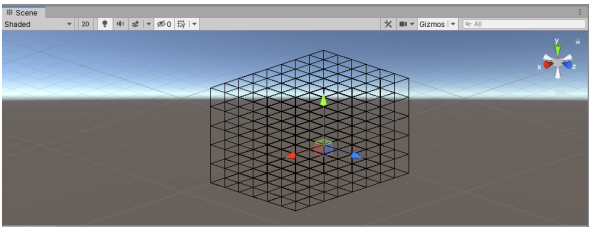


(a) Ukuran = 0.5



(b) Ukuran = 0.7

Gambar 3.3: Perbedaan ukuran voxel memengaruhi ukuran *chunk*.

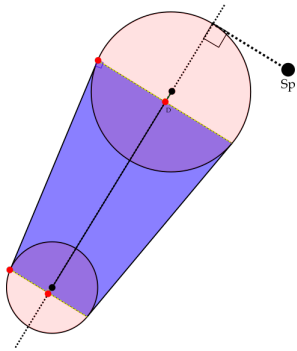


Gambar 3.4: *Chunk* yang sudah dibuat.

3.3.2 Perancangan Garis Jalur Gua

Langkah ini bertujuan untuk merancang dua buah *class* yang akan merepresentasikan sebuah jalur gua beserta bentuk ruangan gua pada tiap jalur. Mengacu artikel oleh Paul Bourke, bangun ruang yang digunakan adalah frustum kerucut [10]. Bangun ruang kapsul tidak digunakan karena muncul permasalahan pemetaan tutup kapsul yang memengaruhi sampel volume. Masalah pemetaan ini juga harus menyesuaikan pemetaan pada selimut kapsul sehingga didapatkan kontinuitas *noise* yang sempurna dari permukaan ujung kapsul ke permukaan selimut kapsul. Frustum kerucut sendi-

ri dapat dianggap sebagai selimut kapsul apabila terdapat kesamaan radius kedua ujung frustum kerucut dan tidak lain adalah bidang tiga dimensi yang bersinggungan dengan permukaan dua bola. Dari pernyataan ini, perhitungan vektor tangent terhadap dua lingkaran dapat digunakan untuk membantu pelaksanaan tahap keempat dari alur kerja dan sebagai dasar variabel apa saja yang diperlukan untuk tahap berikutnya.



Gambar 3.5: Ruang dari setiap jalur gua yang ditampilkan pada bidang biru.

Satu garis jalur dinyatakan oleh dua *class*, yaitu *Node* dan *Line*. Pada *class Node*, terdapat dua variabel, yaitu posisi dan radius, masing-masing dengan tipe data *float3* (*array float* dengan tiga elemen) dan *float*. Kemudian untuk *class Line*, yang menyatakan properti dari satu garis jalur gua, terdapat variabel *nodeA* dan *nodeB* yang menyatakan satu objek *Line* tersambung dengan objek *Node* pada indeks ke berapa. *rightLocal* dan *upLocal* menyatakan vektor unit dari sumbu lokal X dan Y, sedangkan sumbu lokal Z didefiniskan dengan vektor unit dari kedua *Node* yang tersambung. Ketiga sumbu lokal akan digunakan ketika *coordinate space* yang digunakan untuk input *noise* menggunakan *local space coordinate*. Variabel AC menyatakan jarak terdekat antara *Node* tersambung dengan nilai radius terkecil dan titik tengah dari ujung tabung dengan radius terkecil(dinyatakan dalam variabel CE), sedangkan BD menyatakan jarak terdekat antara *Node* tersambung dengan nilai radius terbesar dan titik tengah dari ujung tabung dengan radius

terbesar(dinyatakan dalam variabel *DF*). Terakhir, *lineLength* menyatakan panjang dari garis. Perhitungan untuk semua variabel dilakukan pada aplikasi Unity untuk menghemat waktu eksekusi *compute shader* dalam memproses data volume.

Class LineGenerator bertugas untuk mengatur proses dari pembentukan jalur gua. *lines* dan *nodes* menyimpan objek-objek dari *class Line dan Node*. *activeLineIndex* dan *activeNodeIndex* menyatakan objek *Line* dan *Node* aktif yang bisa diubah propertinya, seperti nilai radius (diubah dalam variabel *activeNodeRadius*) dan posisi objek *Node* (disimpan dalam variabel *activeNodePosition*), serta besar rotasi pada sumbu lokal Z (disimpan dalam *activeLineRotation*). Kemudian terdapat beberapa *method* untuk mengubah data-data dari objek *Node* dan *Line*. *AddLine* dan *ConnectLine* sama-sama memiliki fungsi yang sama, yaitu menambahkan objek *Line* baru, namun dengan cara yang berbeda. *AddLine* menambahkan objek *Node* baru dengan input berupa posisi kursor, kemudian menambahkan objek *Line* baru dengan menyambungkan *Node* elemen ke-*activeNodeIndex* dengan objek *Node* baru, sedangkan *ConnectLine* menambahkan objek *Line* baru dengan menyambungkan kedua objek *Node* yang sudah dibuat.

Setelah menambahkan objek *Line* baru, *CalculateLocalZX*, *RotateLocalX*, dan *CalculateY* dijalankan untuk mencari sumbu lokal XYZ objek *Line*. *CalculateLocalZX* mencari sumbu lokal Z terlebih dahulu dengan mencari vektor unit dari kedua objek *Node* yang tersambung dalam satu objek *Line*. Untuk mencari vektor unit dari sumbu lokal X sempat terdapat kesulitan, yaitu nilai vektor unit seperti apa yang dapat dianggap sebagai rotasi derajat ke 0°. Kami sederhanakan dengan pertama mengubah nilai titik Y pada vektor unit sumbu lokal Z menjadi 0, kemudian merotasi vektor tersebut sebesar 90°searah jarum jam. Vektor unit dari perhitungan sebelumnnya digunakan sebagai sumbu lokal X. Sebelum menghitung sumbu lokal Y, kami merotasi sumbu lokal X terhadap sumbu lokal Z pada *method RotateLocalX* yang melakukan perhitungan *quaternion* untuk merotasi suatu vektor terhadap suatu vektor lain dengan besar rotasi yang didefinisikan pada variabel *activeLineRotation*. Kemudian, sumbu lokal Y didapatkan dengan melakukan perhitungan *cross product* dari vektor unit sumbu lokal Z dan X.

3.3.3 Perancangan Input Noise

Pada umumnya, terdapat beberapa parameter yang dapat memengaruhi hasil dari fungsi *noise*. Parameter-parameter tersebut yaitu *seed*, *octave*, *persistence*, dan *lacunarity*. Parameter *octave* menentukan berapa kali iterasi fungsi *noise* dijalankan. Dalam setiap iterasi, nilai pada parameter *frequency* dan *amplitude* akan terus berubah berdasarkan nilai parameter *persistence* dan *lacunarity*. *Seed* digunakan untuk mendapatkan variasi *noise* lain namun tetap menggunakan nilai-nilai parameter lainnya. Beberapa pengaplikasian *OpenSimplex noise* menyebutkan *seed* dapat digunakan untuk mendapatkan ragam vektor gradien di masing-masing titik petak *simplex* yang berbeda. Penelitian ini hanya menggunakan *seed* untuk menghasilkan sebuah vektor *offset* untuk input posisi *OpenSimplex noise*. Output dari fungsi *OpenSimplex noise* berupa bilangan bulat antara 0 dan 1. Dalam bentuk persamaan, perhitungan *noise* dinyatakan dengan persamaan berikut:

$$noiseVal = \frac{\sum_{o=1}^{octave} noise((s\vec{P}os + off\vec{f}set + oct\vec{O}ff_o) * L^{o-1}) * P^{o-1}}{\sum_{o=1}^{octave} P^{o-1}}$$
$$0 < L < 1, L \in \mathbb{Q}$$
$$P \in \mathbb{R} \tag{3.3}$$

Berikut *pseudocode* untuk mendapatkan nilai *noise*:

```
int octave;
int seed;
float lacunarity;
float persistence;
float amplitude = 1;
float frequency = 1;
float3 input;
float3 offset;
float value = 0;
float maxVal = 0;
var rng = new System.Random(seed);
float3 noiseOffset[8];

for(i=0; i<octave; i++)
{
```

```
    noiseOffset[i] = new float3(rng.Next(), rng.Next(), rng
        .Next());
    value = noise((input+offset+noiseOffset[i])*frequency)*
        amplitude;
    maxVal +=amplitude;
    amplitude *= persistence;
    frequency *= lacunarity;
  }
  value = value / maxVal;
  return value;
```

3.3.4 Perhitungan Titik Sampel yang Terjangkau Garis

Langkah ini bertujuan untuk mendapatkan titik sampel mana saja yang berada dalam ruang gua yang berbentuk frustum kerucut. Perhitungan garis tangent terhadap dua lingkaran membantu penentuan syarat-syarat yang harus dipenuhi sebuah titik sampel agar dianggap terjangkau oleh sebuah jalur gua. Dengan gambar 3.10 sebagai vektor-vektor yang menyusun sebuah garis tangent, sebuah titik sampel ***Sp*** dianggap terjangkau oleh objek *Line* apabila memenuhi syarat berikut:

1. Panjang proyeksi orthogonal titik sampel ***Sp*** terhadap garis pada objek *Line* (***p***) memenuhi syarat:

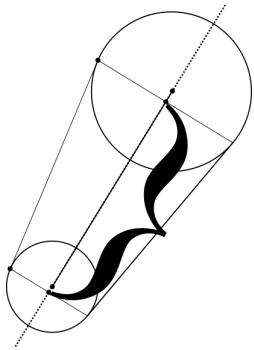
$$-|\vec{AC}| < p < l - |\vec{BD}| \tag{3.4}$$

   di mana AC adalah vektor antara titik tengah ujung tabung tangent dengan panjang terkecil dan titik ***A***, sedangkan *BD* adalah vektor antara titik tengah ujung tabung tangent dengan panjang terbesar dan titik ***B***.

2. Jika kondisi pertama terpenuhi, jarak ***Sp*** dengan objek *Line* tidak lebih dari nilai ***lr*** yang nilainya diperoleh dari:

$$lr = \vec{CE} + \left(\frac{p + |\vec{AC}|}{l + |\vec{AC}| - |\vec{BD}|} * (\vec{DF} - \vec{CE}) \right) \tag{3.5}$$

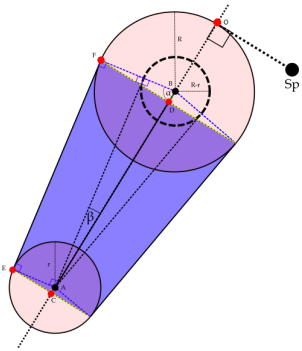
   di mana *lr* adalah interpolasi nilai *CE* dan *DF* berdasarkan nilai *p* pada vektor antara kedua titik tengah ujung frustum



Gambar 3.6: Rentang yang dimaksud pada syarat pertama.

kerucut.

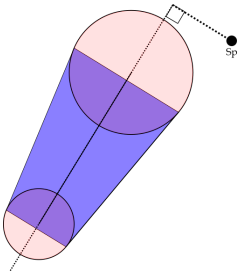
Gambar 3.8 menjelaskan maksud yang ingin dicapai dari kedua syarat tersebut. Pemeriksaan persyaratan dilakukan sesuai dengan jumlah *Line* yang dibuat.



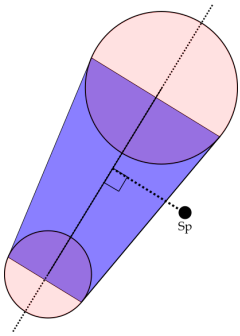
Gambar 3.7: Rincian vektor yang menyusun frustum kerucut.

Langkah pertama adalah menentukan panjang proyeksi titik sampel terhadap garis pada objek *Line* dengan rumus berikut:

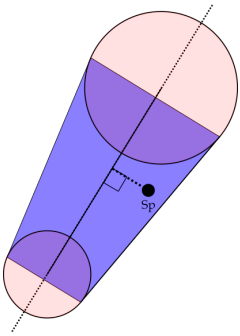
$$p = \frac{\vec{AS}p * \vec{AB}}{|\vec{AB}|} \tag{3.6}$$



(a) Proyeksi titik sampel ke jalur gua di luar bidang trapesium. Kondisi pertama tidak terpenuhi

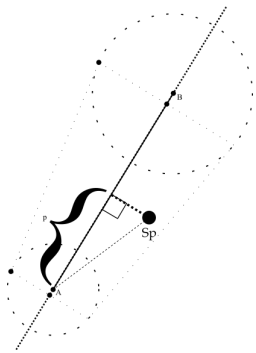


(b) Proyeksi titik sampel ke jalur gua berada di dalam bidang trapesium, namun titik sampel berada di luar trapesium. Kondisi pertama terpenuhi, namun kondisi kedua tidak terpenuhi



(c) Baik titik sampel maupun proyeksinya berada di bidang trapesium. Kedua kondisi terpenuhi.

Gambar 3.8: Tiga kemungkinan kondisi titik sampel berdasarkan lokasinya terhadap sebuah jalur gua.



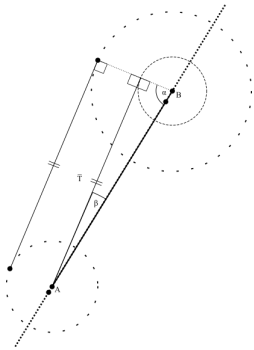
Gambar 3.9: Proyeksi titik sampel *Sp* ke objek *Line*.

Langkah selanjutnya adalah mencari panjang vektor yang bersinggungan dengan kedua lingkaran yang bertitik pusat pada kedua objek *Node* ($\boldsymbol{T}$). Vektor T adalah vektor yang memotong titik A dan bersinggungan dengan bola ketiga, di mana posisinya berada pada titik B dan radiusnya adalah selisih dari kedua nilai radius pada masing-masing objek *Node* yang terhubung. Vektor T, vektor antara dua *Node*, dan vektor antara titik singgung vektor T dengan bola ketiga dan titik B membentuk sebuah segitiga siku-siku, dengan vektor T sebagai hipotenusa, yang akan memudahkan proses pencarian panjang vektor AC, BD, CE, dan DF.

Berdasarkan aturan trigonometri, panjang vektor AC dan BD didapat dengan rumus berikut:

$$\begin{aligned} |\vec{AC}| &= r_A * \sin\beta \\ |\vec{BD}| &= r_B * \sin\beta \\ \sin\beta &= \frac{r_B - r_A}{|\vec{AB}|} \end{aligned} \tag{3.7}$$

Setelah mengetahui AC dan BD, dilakukan pemeriksaan apakah p memenuhi persamaan 3.4. Apabila tidak, titik sampel dianggap tidak terjangkau objek *Line*. Sebaliknya, pemeriksaan untuk syarat kedua dilakukan. Dengan teorema Pythagoras, panjang vek-



Gambar 3.10: Vektor T sejajar dengan vektor tangent terhadap dua lingkaran.

tor CE dan DF didapat dengan rumus berikut:

$$\begin{aligned} |\vec{CE}| &= \sqrt{r_A^2 - |\vec{AC}|^2} \\ |\vec{DF}| &= \sqrt{r_B^2 - |\vec{BD}|^2} \end{aligned} \tag{3.8}$$

Panjang lr dapat diketahui dengan persamaan 3.5. Apabila nilai p tidak memenuhi syarat kedua, titik sampel dianggap tidak terjangkau objek *Line*. Sebaliknya, titik sampel dianggap terjangkau oleh objek *Line*. Apabila dari titik sampel tidak mampu dijangkau oleh seluruh objek *Line*, maka data volume titik sampel secara otomatis bernilai -1 dan tidak dilibatkan dalam perhitungan data volume menggunakan *noise* [9].

Berikut adalah *pseudocode* dari algoritma untuk memeriksa apakah titik sampel terjangkau oleh objek *Line*:

```
List<Line> _lines;
List<Node> _nodes;
Line closestLine;
Node closestNodeA, closestNodeB;
Node nA, nB;
float lr, p, d, AC, BD, CE, DF;
float3 AS;
float4 samplePoint;
int closestLineIndex = -1;
```

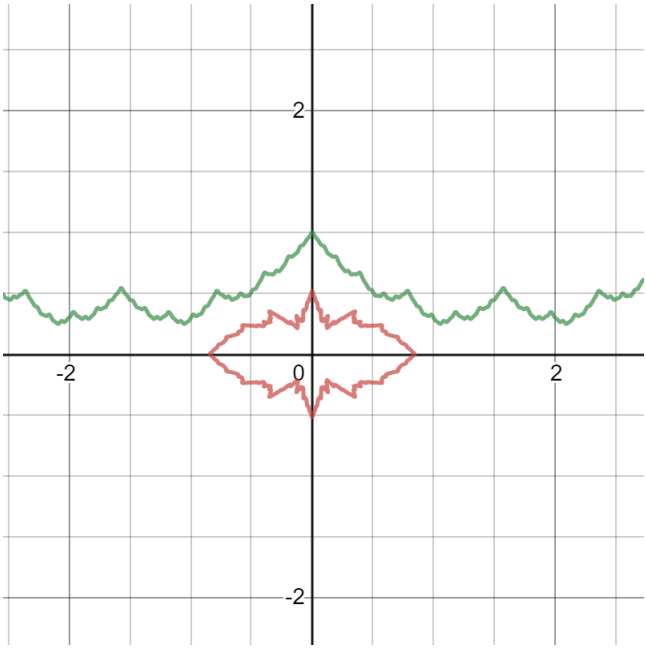
```
for(int i=0; i<_lines.Count; i++)
{
  if(_nodes[_lines.node1].radius < _nodes[_lines.node2].
      radius)
  {
    nA = _nodes[_lines[i].node1].radius;
    nB = _nodes[_lines[i].node2].radius;
  }
  else
  {
    nA = _nodes[_lines[i].node2].radius;
    nB = _nodes[_lines[i].node1].radius;
  }
  AS = samplePoint.xyz - nA.position;
  p = dot(AS, normalize(nB.position-nA.position));
  AC = nA.radius * (nB.radius - nA.radius)/_lines[i].
      lineLength;
  BD = nB.radius * (nB.radius - nA.radius)/_lines[i].
      lineLength;

  if(p > -AC && p < _lines[i].lineLength - BD)
  {
    CE = sqrt(pow(nA.radius,2)-pow(AC,2));
    DF = sqrt(pow(nB.radius,2)-pow(BD,2));
    lr = (p+AC) / (_lines[i].lineLength + AC - BD) * (DF
        - CE) + CE;
    d = sqrt(pow(magnitude(AS),2)-pow(p,2));
    if(d <= lr)
    {
      closestLineIndex = i;
    }
  }
}

closestLine = _lines[closestLineIndex];
closestNodeA = _nodes[closestLine.node1];
closestNodeB = _nodes[closestLine.node2];
orthoProject = dot(samplePoint.xyz - closestNodeA.
    position, normalize(closestNodeB-closestNodeA));
vOrthoProject = closestNodeA.position + (orthoProject * (
    normalize(closestNodeB-closestNodeA)));

return closestLineIndex;
```

3.3.5 Perhitungan Nilai *Noise* untuk Mendapatkan Sampel Volume

Langkah ini bertujuan untuk menghasilkan sebuah sampel volume yang digunakan untuk menghasilkan model 3D gua. Volume gua akan terbagi menjadi dua daerah, yaitu daerah jalan dan daerah tembok atau langit-langit gua. Tujuan ini dicapai dengan menghasilkan sebuah *noise*, kemudian memproyeksikan *noise* dalam koordinat polar dengan mengubah sistem koordinat yang digunakan untuk input *noise*. Di antara vektor yang menghubungkan titik sampel dengan jalur gua dan vektor sumbu X lokal jalur gua didapatkan sebuah sudut yang digunakan sebagai input posisi pada sumbu X, sedangkan panjang proyeksi orthogonal titik sampel ke jalur gua digunakan sebagai input posisi pada sumbu Y. Sebuah perhitungan *falloff* dilakukan untuk mendapatkan volume ruang gua dan kemudian ditampung dalam vertex voxel. Sampel volume yang berbeda-beda pada tiap titik sampel diperlukan pada tahap interpolasi vertex *sub-mesh* pada proses *marching cube*. Gambar 3.12 merupakan hasil akhir yang ingin dicapai dari tahap ini dengan daerah hitam adalah daerah di bawah permukaan dan daerah putih adalah daerah di atas permukaan.

Apabila titik sampel terjangkau oleh objek *Line*, nilai volume dari titik sampel dapat diperoleh dari fungsi *noise*. Fungsi *noise* yang digunakan adalah fungsi *OpenSimplex noise* yang terbukti menghabiskan waktu komputasi yang lebih cepat daripada fungsi *Perlin noise* [18]. Dimensi dari koordinat untuk input *noise* menggunakan dimensi dua, dengan komponen x adalah nilai sudut antara vektor titik sampel dengan objek *Line* dan vektor sumbu lokal X dan komponen y adalah panjang proyeksi orthogonal titik sampel terhadap objek *Line*.

$$\begin{aligned}
\vec{A} &= \vec{SP} - \vec{AO} \\
\vec{B} &= X_{local} \\
\theta &= \arccos \frac{\vec{A} \cdot \vec{B}}{|\vec{A}| * |\vec{B}|} \\
\vec{input} &= \{\theta/360^\circ, \frac{p}{l}\}
\end{aligned} \tag{3.9}$$

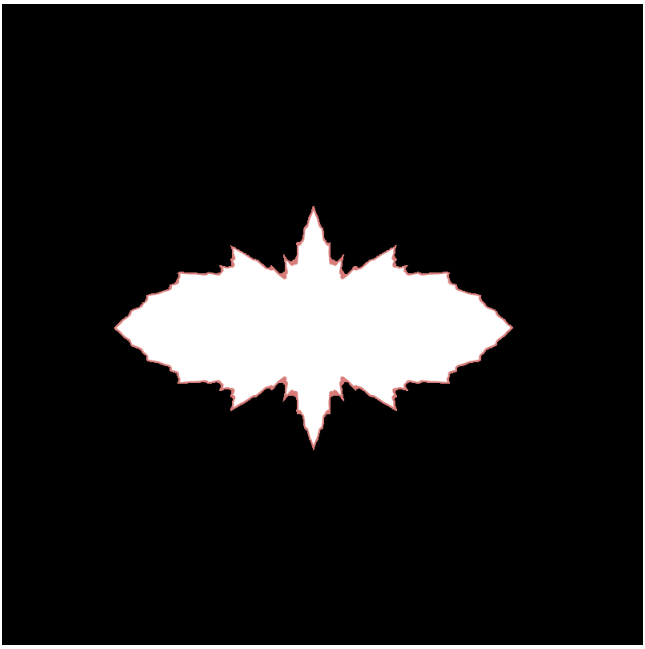


Gambar 3.11: Representasi *noise* pada koordinat polar.

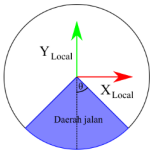
Nilai *noise* yang didapat dari persamaan 3.3.3 digunakan untuk mengubah nilai *lr*. Semakin besar nilai *noise*, nilai *lr* akan semakin mengecil. Seberapa besar pengaruh nilai *noise* bergantung pada daerah mana titik sampel berada. Apabila titik sampel berada pada daerah jalan, maka intensitas *noise* akan mengecil. Daerah jalur kami atur pada rentang sudut antara 225°hingga 315°. Perubahan nilai *lr* berdasarkan nilai *noise* yang didapat dinyatakan dalam persamaan berikut:

$$nDist = \begin{cases} lr - (\frac{noiseVal*lr}{2}), & \text{simulatePathway} = \text{false} \\ lr - (\frac{noiseVal*lr}{8}), & \text{simulatePathway} = \text{true} \end{cases} \tag{3.10}$$

Nilai volume titik sampel diperoleh berdasarkan jarak terdekat

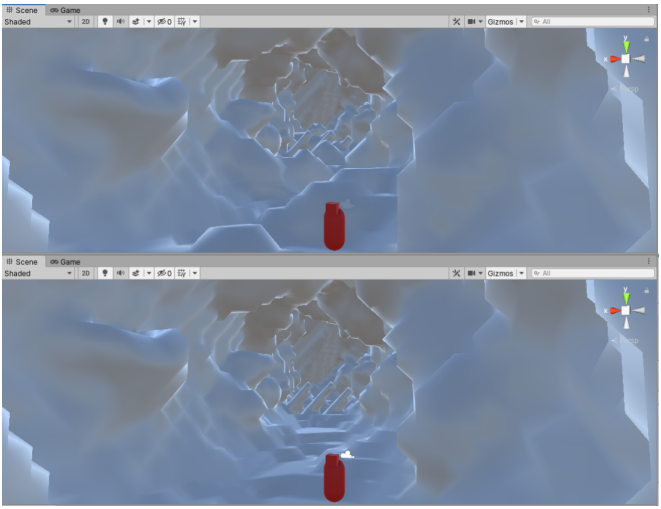


Gambar 3.12: Representasi volume ruang gua dari gambar 3.11 sebelum dilakukan perhitungan *falloff*.



Gambar 3.13: Daerah jalan yang ditunjukkan pada bagian berwarna biru.

titik sampel ke objek *Line* (*dist*) terhadap nilai *nDist*. Apabila *dist* lebih kecil dari nilai *nDist*, secara otomatis nilai volume bernilai *iso level*, di mana nilai *iso level* adalah nilai *threshold* yang menentukan apakah titik sampel berada di bawah permukaan atau di atas permukaan. Sebaliknya, nilai volume adalah nilai *falloff* dengan nilai *nDist* ke nilai *lr* sebagai radius minimal dan radius maksimal. Hasil



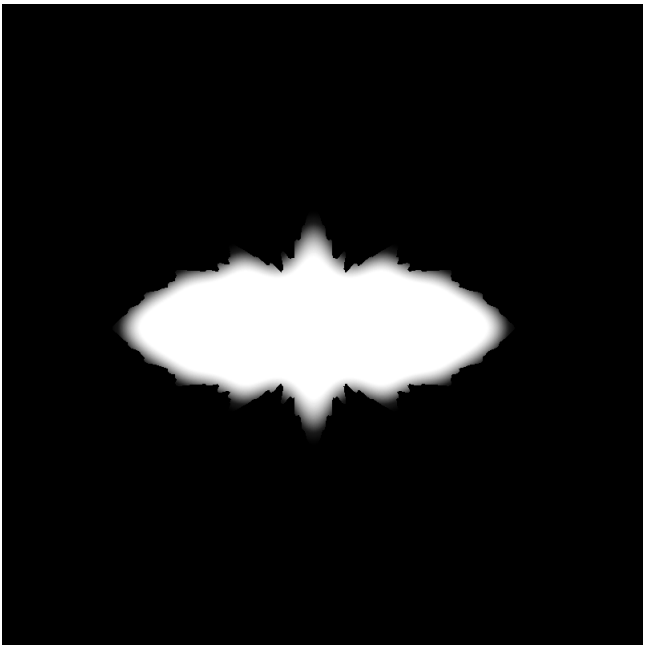
Gambar 3.14: Permukaan gua tanpa daerah jalan dan dengan daerah jalan.

akhir dari perhitungan ini terdapat pada gambar 3.15.

$$
\begin{aligned}
volume &= falloff(dist, nDist, lr) \\
falloff(r, r_{min}, r_{max}) &= clamp\left(\frac{r - r_{min}}{r_{max} - r_{min}}, 0, 1\right)
\end{aligned}
\tag{3.11}
$$

Berikut adalah *pseudocode* untuk perhitungan data volume menggunakan fungsi *OpenSimplex noise*:

```
float angle, dist, interR, orthoProject, noiseVal, nDist,
     volumeSample;
float2 input;
float3 a, b, vOrthoProject;
float4 samplePoint;
bool simulatePathway;
bool isInPathZone = false;

dist = length(samplePoint.xyz - orthoProject);
interR = closestLine.CE + ((orthoProject+closestLine.AC)
    /(closestLine.lineLength+closestLine.AC-closestLine.
    BD)*(closestLine.DF-closestLine.CE));
```



Gambar 3.15: Contoh sampel volume ruang gua.

```
a = samplePoints[index].xyz - orthoProject;
        b = closestLineProps.xLocal;// + orthoProject;
        yLocal = dot(a,closestLineProps.yLocal);
        angle = degrees((acos(dot(a, b)/(length(a)*
            length(b)))));
        if(yLocal<0)
        {
            angle = 360 - angle;
            if(simulatePathway)
            {
             float leftAngleBorder, rightAngleBorder;
             leftAngleBorder = 225;
             rightAngleBorder = 315;
             if(angle > leftAngleBorder && angle <
                 rightAngleBorder)
             {
               isInPathZone = true;
             }
```

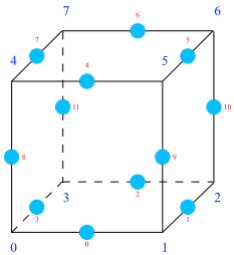
```
                }
            }
            angle \%= 360;

            input.x = angle/360;
            input.y = orthoProject / closestLine.lineLength;

            noiseVal = OpenSimplex(input);
            if(isInPathZone)
            {
              nDist = interR - (noiseVal * interR / 8);
    }
    else
    {
      nDist = interR - (noiseVal * interR / 2);
    }
            volumeSample = falloff(dist, noiseDist, interR);
                 //falloff(r, r_min, r_max)
            return volumeSample;
```

3.3.6 Rekonstruksi Model 3D

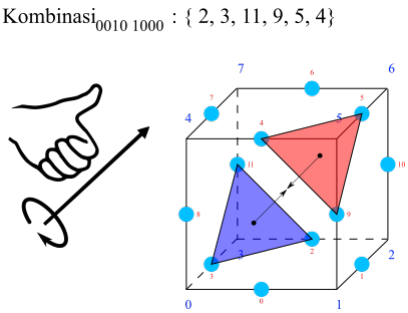
Data volume yang didapat dari seluruh titik sampel akan diubah menjadi model 3D dengan menggunakan algoritma *marching cube*. Algoritma ini digunakan karena sering digunakan baik untuk penelitian maupun pengaplikasian *procedural environment generation*, namun terdapat kekurangan yaitu memungkinkan munculnya artifak melayang pada model 3D yang dihasilkan [19][20]. Hal ini dapat diatasi dengan mengubah resolusi dari voxel yang digunakan sehingga data volume yang didapat menjadi lebih rinci. Hasil dari algoritma ini adalah model 3D yang umumnya memiliki properti-properti seperti *vertex buffer* dan *index buffer*. Kedua properti ini dapat diubah dalam sebuah *struct* yang disebut sebagai *Triangle* yang memiliki tiga variabel dengan tipe data *float3* untuk menyimpan posisi-posisi dari ketiga vertex yang menyusun sebuah *face/ triangle*.

*S*ejumlah titik-titik baru ditambahkan di tengah-tengah setiap rusuk voxel yang menjadi kandidat vertex dari *sub-mesh* yang akan dihasilkan. Titik baru ini diberi sebuah indeks yang mengikuti aturan pemberian indeks pada gambar 3.16. Sebuah *array* dibuat untuk menampung kemungkinan *sub-mesh* yang akan dibuat di tiap voxel. *Array* ini disebut sebagai *march table* di mana setiap elemennya me-

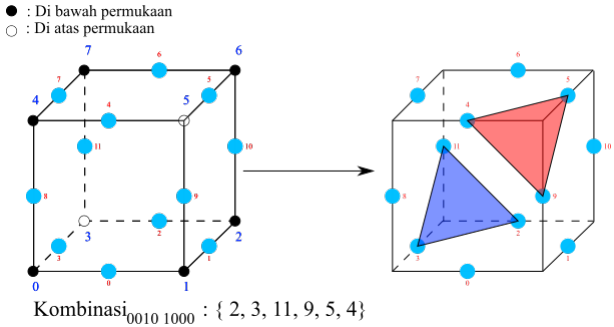
nampung *index buffer sub-mesh*. *Index buffer* berisi indeks dari titik tengah yang telah dibuat. Sampel volume yang dimiliki oleh setiap vertex voxel akan dibandingkan oleh suatu nilai *threshold* yang disebut sebagai *iso level*. Dari *thresholding* ini, vertex akan memiliki dua kondisi; berada di atas permukaan apabila sampel volume lebih atau sama dengan dari nilai *iso level* dan berada di bawah permukaan apabila sampel volume kurang dari nilai *iso level*. Dengan begitu, terdapat 256 kombinasi kondisi vertex voxel yang akan memengaruhi bentuk *sub-mesh* yang dihasilkan. Apabila titik tengah rusuk voxel berada di antara kedua vertex voxel dengan kondisi berbeda, maka posisi titik tengah tersebut dimasukkan dalam *vertex buffer sub-mesh* dan indeks dari elemen *vertex buffer* yang telah ditambahkan tadi disertakan dalam *index buffer sub-mesh*. Perlu diperhatikan bahwa urutan penulisan indeks vertex pada *index buffer* berpengaruh terhadap arah vektor normal *face/ triangle sub-mesh*. Vektor normal yang benar dari sebuah *face/ triangle* mengarah dari permukaan menuju daerah di atas permukaan. Kesalahan urutan penulisan dapat menyebabkan arah vektor normal mengarah ke arah yang salah dan dalam beberapa kondisi sebuah *face/ triangle* tidak akan ter-*render* karena pengaturan *backface culling* sebuah kamera baik pada *game engine* maupun aplikasi pengolah model 3D. Kaidah tangan kiri dapat digunakan untuk memudahkan pengurutan penulisan indeks vertex yang dijelaskan pada gambar 3.17 dengan bidang biru adalah bidang yang tidak menerima dampak *backface culling* dan bidang merah adalah bidang yang menerima dampak *backface culling*. Kombinasi kondisi kedelapan vertex voxel akan



Gambar 3.16: Pemberian indeks pada vertex voxel dan titik tengah rusuk voxel.



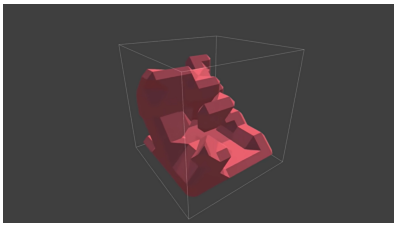
Gambar 3.17: *Sub-mesh* yang dihasilkan pada kombinasi 40.



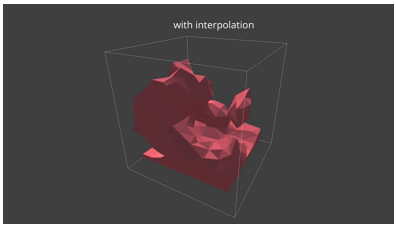
Gambar 3.18: Proses *marching cube* setelah membandingkan sampel volume dengan nilai *iso level*.

menghasilkan sebuah *index buffer* dari *march table* yang dapat digunakan untuk memproses posisi titik tengah rusuk voxel. Titik ini akan diinterpolasi berdasarkan sampel volume dari dua vertex voxel yang terhubung agar didapatkan bentuk model 3D gua yang sesuai dengan *noise* pada tahap sebelumnya. Nilai interpolasi posisi inilah yang digunakan sebagai posisi dari vertex *sub-mesh* untuk model 3D gua yang kemudian digunakan sebagai input untuk membuat objek *Triangle* baru.

$$
\begin{aligned}
i &= \frac{isoLevel - volSample_1}{volSample_2 - volSample_1} \\
pos' &= pos_1 + (i * (pos_2 - pos_1))
\end{aligned}
\tag{3.12}
$$



(a) Hasil model 3D tanpa proses interpolasi.



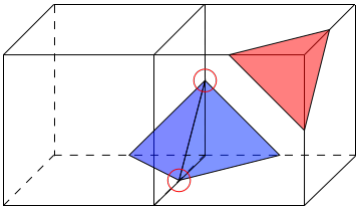
(b) Hasil model 3D dengan proses interpolasi.

Gambar 3.19: Perbandingan model 3D tanpa proses interpolasi dengan model 3D yang sudah diinterpolasi [1].

Vertex-vertex dari setiap *sub-mesh* model 3D yang dihasilkan tertumpuk dengan vertex lain. Hal ini menandakan bahwa terdapat beberapa vertex berada pada posisi yang sama. Untuk mengatasi masalah ini, operasi *vertex welding* sederhana dapat dilakukan untuk menggabungkan vertex-vertex dengan posisi yang sama serta mengurangi jumlah vertex pada model 3D. Terakhir, vektor normal vertex perlu dihitung ulang agar saat objek di-*render*, objek akan terlihat lebih halus [14]. Operasi ini telah disediakan oleh *game engine Unity* pada fungsi *Mesh.RecalculateNormals()*.

Berikut adalah *pseudocode* untuk algoritma *marching cube*:

```
#include "Include/MarchTables.compute"  //berisi march
    table
#include "Include/Inputs.compute"   //input

struct Triangle {
    float3 vertexC;
    float3 vertexB;
    float3 vertexA;
```



Gambar 3.20: Keempat vertex yang tertumpuk pada dua posisi berbeda.

```
};

AppendStructuredBuffer<Triangle> triangles; //buffer dengan
     ukuran dinamis untuk menyimpan triangle model 3D gua

float4 vertices[8];

void GetVertices(uint3 id)
{
    uint firstVertIndex = id.x + (id.z * numVertexX) + (id.
        y * numVertexX * numVertexZ);

    vertices[0] = samplePoints[firstVertIndex];
    vertices[1] = samplePoints[firstVertIndex+1];
    vertices[2] = samplePoints[firstVertIndex+1+numVertexX
        ];
    vertices[3] = samplePoints[firstVertIndex+numVertexX];
    vertices[4] = samplePoints[firstVertIndex+(numVertexX*
        numVertexZ)];
    vertices[5] = samplePoints[firstVertIndex+(numVertexX*
        numVertexZ)+1];
    vertices[6] = samplePoints[firstVertIndex+(numVertexX*
        numVertexZ)+1+numVertexX];
    vertices[7] = samplePoints[firstVertIndex+(numVertexX*
        numVertexZ)+numVertexX];
}

//periksa kondisi titik sampel
bool CheckDensity(float4 samplePoint)
{
    if(samplePoint.w>=isoLevel)
    {
        return true;
    }
```

```
    else
    {
        return false;
    }
}

//interpolasi posisi titik tengah rusuk voxel
float3 interpolateVerts(float4 v1, float4 v2) {
    float t = (isoLevel - v1.w) / (v2.w - v1.w);
    return v1.xyz + (t*(v2.xyz-v1.xyz));
}

[numthreads(1,1,1)]
void GenerateMesh (uint3 id : SV_DISPATCHTHREADID)    //id
    = index kubus
{
    //dari dispatch thread yang aktif, dapatkan indeks
        vertex
    //indeks vertex yang didapat digunakan sebagai vertex
        indeks ke-0
    GetVertices(id);
    int combinationIndex=0;

    if(CheckDensity(vertices[0])) combinationIndex |= 1;
    if(CheckDensity(vertices[1])) combinationIndex |= 2;
    if(CheckDensity(vertices[2])) combinationIndex |= 4;
    if(CheckDensity(vertices[3])) combinationIndex |= 8;
    if(CheckDensity(vertices[4])) combinationIndex |= 16;
    if(CheckDensity(vertices[5])) combinationIndex |= 32;
    if(CheckDensity(vertices[6])) combinationIndex |= 64;
    if(CheckDensity(vertices[7])) combinationIndex |= 128;

  //proses pembentukan triangle dari kombinasi yang didapat
    for(int i=0; triangulation[combinationIndex][i]!= -1; i
        +=3)
    {
        //cornerIndexAFromEdge dan cornerIndexBFromEdge
            adalah fungsi untuk mendapatkan kedua vertex
            yang mengapit titik tengah rusuk voxel
        int a0 = cornerIndexAFromEdge[triangulation[
            combinationIndex][i]];
        int b0 = cornerIndexBFromEdge[triangulation[
            combinationIndex][i]];

        int a1 = cornerIndexAFromEdge[triangulation[
            combinationIndex][i+1]];
        int b1 = cornerIndexBFromEdge[triangulation[
```

```
            combinationIndex][i+1]];

        int a2 = cornerIndexAFromEdge[triangulation[
            combinationIndex][i+2]];
        int b2 = cornerIndexBFromEdge[triangulation[
            combinationIndex][i+2]];

        Triangle tri;
        tri.vertexA = interpolateVerts(vertices[a0],
            vertices[b0]);
        tri.vertexB = interpolateVerts(vertices[a1],
            vertices[b1]);
        tri.vertexC = interpolateVerts(vertices[a2],
            vertices[b2]);
        triangles.Append(tri);
    }
}
```

BAB 4
PENGUJIAN DAN ANALISIS

Pengujian dilakukan pada sebuah laptop dengan spesifikasi sebagai berikut:

1. *Operating system*: Windows 10 64-bit
2. RAM: 4 GB
3. CPU: Intel Core i5-7200u
4. GPU: Nvidia GeForce 940MX
5. *Graphics API*: Direct3D11
6. Ter-*install game engine* Unity versi 2019.3

Sedangkan untuk jenis pengujian yang dilakukan dibagi menjadi lima macam:

1. Perubahan resolusi voxel
2. Perubahan paramater *noise*
3. Perubahan jumlah jalur gua
4. Rekonstruksi kelima jenis gua
5. Survei kepuasan pengguna

Pengujian dilakukan untuk mengamati pengaruh perubahan parameter input terhadap waktu pembuatan model 3D, jumlah vertex dan *triangle* model, dan karakteristik model 3D. Pada semua pengujian, disediakan beberapa objek *Node* sesuai pada tabel 4.1 dan beberapa objek *Line* pada tabel 4.2. Selain disajikan dalam bentuk tabel, hasil pengujian juga akan disajikan dalam gambar dengan me-*render* salah satu bagian model 3D pada *software 3D modelling Blender*.

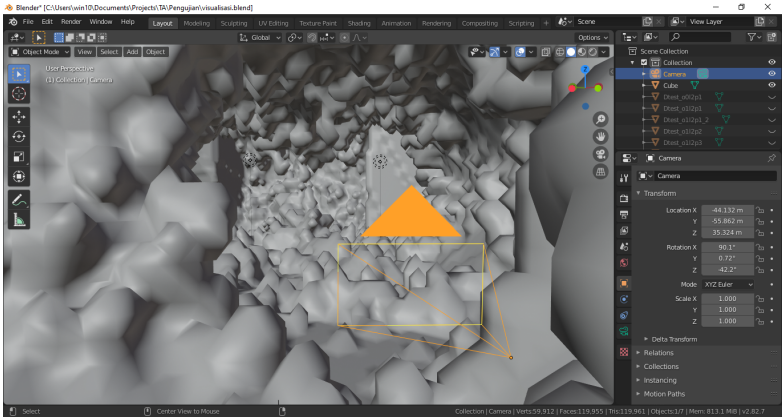
Tabel 4.1: Tabel atribut objek *Node*

Indeks	Posisi (X, Y, Z)	Radius
0	(37.5280, 36.4901, 51.8125)	5.88
1	(73.7031, 45.1403, 45.1712)	14
2	(8.5000, 38.0005, 26.6951)	7.16
3	(17.0716, 38.7969, 9.0205)	7.48
4	(25.1120, 9.3047, 31.4384)	6.61
5	(44.3638, 38.3281, 34.8655)	4.62
6	(74.0255, 9.7109, 33.8975)	5.33
7	(49.0243, 10.6955, 17.6719)	6
8	(72.2983, 8.7188, 8.9599)	7.15

Tabel 4.2: Tabel atribut objek *Line*

Indeks	*Node* A	*Node* B
0	0	1
1	0	2
2	2	3
3	3	4
4	0	3
5	1	5
6	4	6
7	6	7
8	7	8
9	8	3

Semua model 3D gua yang telah dihasilkan pada *Unity* akan di-*export* menjadi file *OBJ* yang kemudian akan di-*import* ke aplikasi *3D modelling Blender*. Setiap model akan di-*render* sebagian lorongnya dengan pengaturan kamera dan pencahayaan pada gambar 4.1. Hasil *render* digunakan untuk pengamatan dan membandingkan karakteristik model 3D gua dengan model 3D gua lainnya yang sudah dibuat.



Gambar 4.1: Konfigurasi kamera dan pencahayaan pada *Blender*.

Setelah itu, program akan diuji untuk merekonstruksi kelima jenis gua, yaitu *branchwork*, *network*, anastomotik, *ramiform*, dan *spongework*. Survei pengguna dilakukan untuk mendapatkan *feedback* mengenai antar muka program dan pendapat pengguna tentang model 3D yang dihasilkan dengan program kami.

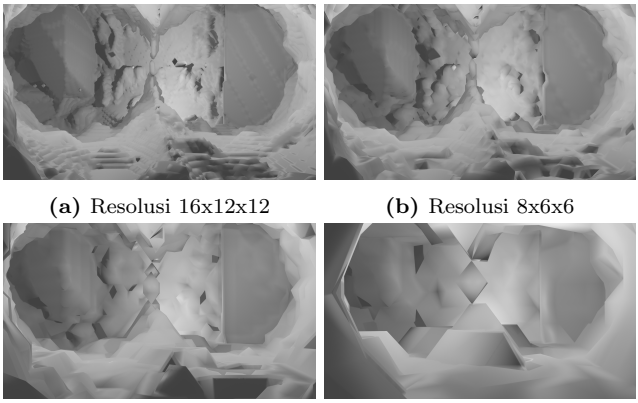
4.1 Perubahan Resolusi Voxel

Pengujian ini dilakukan dengan mengamati banyaknya *chunk* yang digunakan untuk mengambil sampel data volume. Banyaknya *chunk* yang digunakan berubah menurut ukuran resolusi *chunk*. Tabel 4.3 menunjukkan hasil pengujian beserta input berupa resolusi *chunk*.

Tabel 4.3: Tabel pengujian perubahan resolusi voxel.

Pengujian	Input		Output			
	Resolusi (X, Y, Z)	Ukuran voxel	Vertex		*Triangles*	Waktu
			Setelah *vertex welding*	Sebelum *vertex welding*		
1	(16, 12, 12)	0.25	1,956,615	246,720	652,205	20.11
2	(8, 6, 6)	0.5	483,144	58,300	161,048	1.78
3	(4, 3, 3)	1	119,535	13,114	39,845	0.54
4	(2, 3, 3)	2	33,981	2,857	11,327	0.18

Terlihat pada tabel 4.3 bahwa semakin kecil resolusi voxel, semakin singkat waktu yang dibutuhkan untuk membuat model 3D dan jumlah vertex baik sebelum dilakukan proses *vertex welding* ma-



(a) Resolusi 16x12x12 **(b)** Resolusi 8x6x6

(c) Resolusi 4x3x3 **(d)** Resolusi 2x3x3

Gambar 4.2: Perbandingan sebagian bagian model 3D menurut perubahan resolusi voxel.

upun setelahnya beserta *triangle* juga mengecil, namun dari model 3D yang dihasilkan menunjukkan kurangnya detail yang dimunculkan. Perubahan ini dapat diamati pada gambar 4.2.

4.2 Perubahan Parameter Noise

Pengujian ini dibagi menjadi tiga macam, yaitu perubahan nilai *octave*, *persistence*, dan *lacunarity*. Untuk resolusi voxel yang digunakan pada pengujian ini adalah 8x6x6. Untuk mempermudah pengamatan, terutama pada pengujian perubahan *lacunarity* dan *persistence*, kami melakukan simulasi *noise* sederhana pada kalkulator grafik *Desmos* sebagai referensi apa pengaruh *lacunarity* dan *persistence* sebenarnya. Fungsi *noise* yang kami gunakan pada *Desmos* adalah:

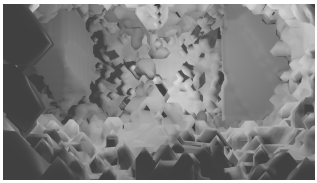
$$\begin{aligned} &\frac{\Sigma_{n=1}^{o} \frac{(\cos(l^n\theta)+1)}{2} * p^n}{\Sigma_{n=1}^{o} l^n} \\ &l \ = Lacunarity \\ &p \ = Persistence \\ &o \ = Octave \end{aligned} \tag{4.1}$$

4.2.1 *Octave*

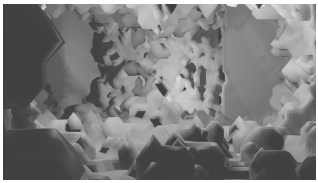
Pengujian ini mengamati pengaruh perubahan nilai *octave* terhadap perubahan bentuk model 3D gua beserta waktu yang diperlukan untuk membuat model 3D. Tabel 4.5 adalah tabel pengujian pada percobaan perubahan parameter input *noise* dan resolusi *chunk* yang digunakan adalah 8x6x6 dengan ukuran voxel sebesar 0,5 sesuai dengan satuan unit pada *game engine Unity*.

Tabel 4.4: Tabel pengujian perubahan parameter *octave* untuk input *noise*.

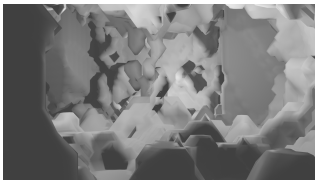
Pengujian	Input				Output			
	Seed	*Octave*	*Lacunarity*	*Persistence*	Vertex		*Triangles*	Waktu
					Sebelum *vertex welding*	Setelah *vertex welding*		
1	15	1	2	0.979	529116	48719	176372	1.1805
2	15	2	2	0.979	546792	50586	182264	1.1373
3	15	3	2	0.979	591120	54141	197040	1.3328
4	15	4	2	0.979	647592	57431	215864	1.2919



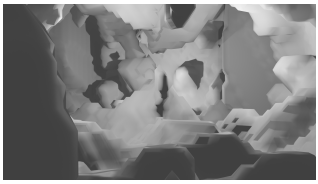
(a) *octave* = 4



(b) *octave* = 3



(c) *octave* = 2



(d) *octave* = 1

Gambar 4.3: Perbandingan sebagian bagian model 3D menurut perubahan nilai *octave*.

Pada *octave* bernilai satu, bagian-bagian bukit terlihat lebih halus, yang kemudian akan terlihat semakin landai dan muncul artifak tajam ketika *octave* semakin besar. Hal ini sesuai dengan pengaplikasian algoritma *noise* pada umumnya. Apabila output *noise* dipetakan dalam sebuah grafik, gelombang *noise* pada *octave* yang lebih kecil terlihat halus. Dengan penggabungan nilai *noise* lain

dengan amplitudo yang semakin kecil dan frekuensi yang semakin besar pada setiap *octave*, gelombang yang dihasilkan akan semakin landai dan mulai muncul beberapa *spike* pada gelombang *noise*.

4.2.2 *Persistence*

Pengujian ini mengamati pengaruh nilai *persistence* terhadap bentuk model 3D gua yang dihasilkan. Sebagai perbandingan, kami melakukan simulasi *noise* pada kalkulator *Desmos* untuk mengamati pengaruh sebenarnya dari masing-masing variabel terhadap *noise* yang dihasilkan. Gambar 4.5 menunjukkan hasil simulasi kami dengan parameter input sebagai berikut:

- $Octave = 7$
- $Lacunarity = 2$
- $Persistence = \{0.99, 0.75, 0.5, 0.25\}$

Tabel 4.5: Tabel pengujian perubahan parameter *persistence* untuk input *noise*.

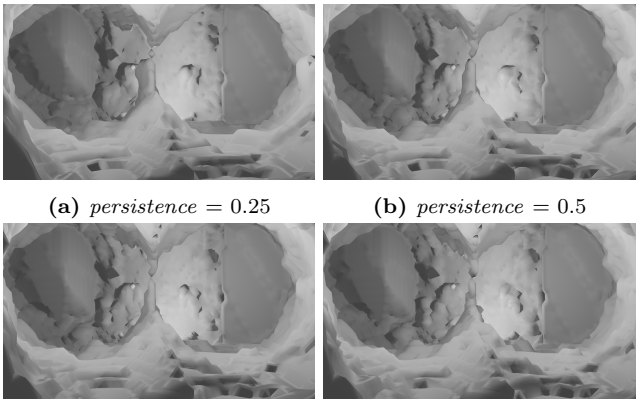
Pengujian	Input				Output			
					Vertex			
	Seed	*Octave*	*Lacunarity*	*Persistence*	Sebelum *vertex welding*	Setelah *vertex welding*	*Triangles*	Waktu
1	15	2	2	0.979	483144	58300	161048	1.1499
2	15	2	2	0.75	480591	58159	160197	1.1304
3	15	2	2	0.5	476721	57914	158907	1.0425
4	15	2	2	0.25	472656	57851	157552	1.1173

Terdapat tiga macam grafik pada gambar 4.5 dengan nilai *persistence* yang berbeda-beda (Biru = 0.979. Hijau = 0.5, Merah = 0.25). Terlihat pada grafik biru bahwa terdapat banyak artifak *spike*. Saat nilai *persistence* semakin kecil, grafik *noise* terlihat semakin halus dan mendekati bentuk grafik persamaan trigonometri pada umumnya, serta beberapa *spike* pada grafik *noise* semakin kecil.

Setelah menjalankan pengujian, kami mendapatkan beberapa hasil yang berbeda dengan hasil simulasi. Hasil pengujian tidak menunjukkan banyaknya *spike* pada nilai *persistence* tinggi. Terdapat perubahan pada beberapa bagian permukaan gua yaitu muncul bagian-bagian menonjol.

4.2.3 *Lacunarity*

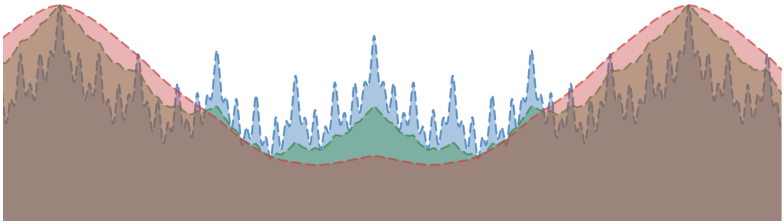
Pengujian ini mengamati pengaruh nilai *lacunarity* terhadap bentuk dari model 3D yang dibuat. Seperti pengujian perubahan



(a) *persistence* = 0.25 **(b)** *persistence* = 0.5

(c) *persistence* = 0.75 **(d)** *persistence* = 0.979

Gambar 4.4: Perbandingan sebagian bagian model 3D menurut perubahan nilai *persistence*.



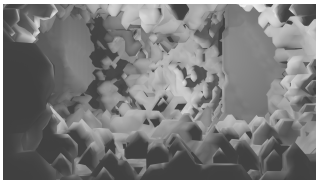
Gambar 4.5: Pengaruh perubahan *persistence* pada simulasi oleh *Desmos*.

nilai *persistence*, simulasi akan dijalankan pada kalkulator *Desmos* untuk mengamati pengaruh sebenarnya dari perubahan nilai *lacunarity*. Hasil simulasi dapat dilihat pada gambar 4.7 dengan parameter input sebagai berikut:

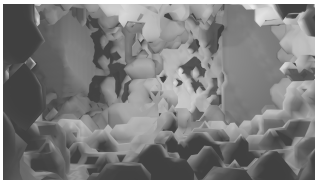
- *Octave* = 2
- *Lacunarity* = {1, 2, 3, 4}
- *Persistence* = 0.979

Tabel 4.6: Tabel pengujian perubahan parameter *lacunarity* untuk input *noise*.

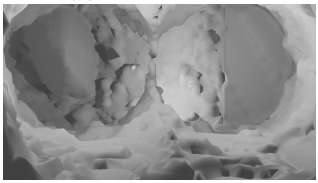
Pengujian	Input				Output			
	Seed	*Octave*	*Lacunarity*	*Persistence*	Vertex		*Triangles*	Waktu
					Sebelum *vertex welding*	Setelah *vertex welding*		
1	15	2	1	0.979	463185	56746	154395	1.5146
2	15	2	2	0.979	546792	50586	182264	1.1373
3	15	2	3	0.979	594720	54814	198240	1.1702
4	15	2	4	0.979	657360	59904	219120	1.3571



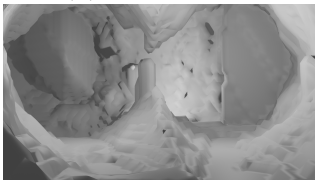
(a) *lacunarity* = 4



(b) *lacunarity* = 3

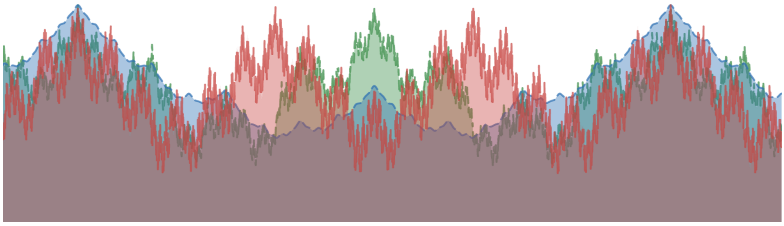


(c) *lacunarity* = 2



(d) *lacunarity* = 1

Gambar 4.6: Perbandingan sebagian bagian model 3D menurut perubahan nilai *lacunarity*.



Gambar 4.7: Pengaruh perubahan *lacunarity* pada simulasi oleh *Desmos*.

Terdapat tiga macam grafik pada gambar 4.5 dengan nilai *persistence* yang berbeda-beda (Biru = 2. Hijau = 4, Merah = 6).

Terlihat pada grafik hijau bahwa terdapat banyak artifak *spike*. Saat nilai *lacunarity* semakin kecil, tinggi *spike* semakin berkurang dan jarak antar puncak gelombang *noise* semakin menjauh. Setelah menjalankan pengujian, terlihat model 3D gua semakin kasar saat nilai *lacunarity* semakin tinggi dan muncul *spike* pada permukaan model 3D gua, namun tidak tampak perubahan jarak puncak pada model 3D seperti pada simulasi dikarenakan sulitnya mengamati perubahan tersebut ketika data volume telah berubah menjadi model 3D.

4.3 Perubahan Jumlah Jalur Gua

Pengujian ini mengamati pengaruh jumlah jalur gua terhadap waktu yang dibutuhkan untuk menghasilkan model 3D gua. Pengujian ini dilakukan dengan mengubah jumlah jalur gua dengan pengujian pertama menggunakan jumlah jalur gua beserta atributnya sesuai dengan tabel 4.1 dan tabel 4.2, kemudian menghilangkan objek *Line* pada indeks terakhir untuk pengujian berikutnya hingga jumlah objek *Line* adalah satu. Hasil pengujian dapat dilihat pada tabel 4.7.

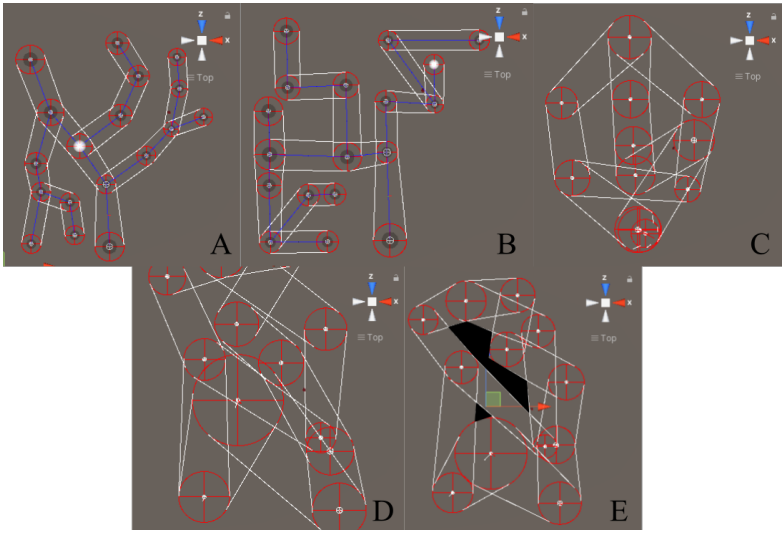
Tabel 4.7: Tabel hasil pengujian perubahan jumlah jalur gua.

Pengujian	Input	Output
	Jumlah jalur gua	Waktu
1	10	1,2071
2	9	1,2226
3	8	1,2114
4	7	1,1953
5	6	1,1998
6	5	1,1654
7	4	0,9170
8	3	1,0994
9	2	1,1151
10	1	0,0891

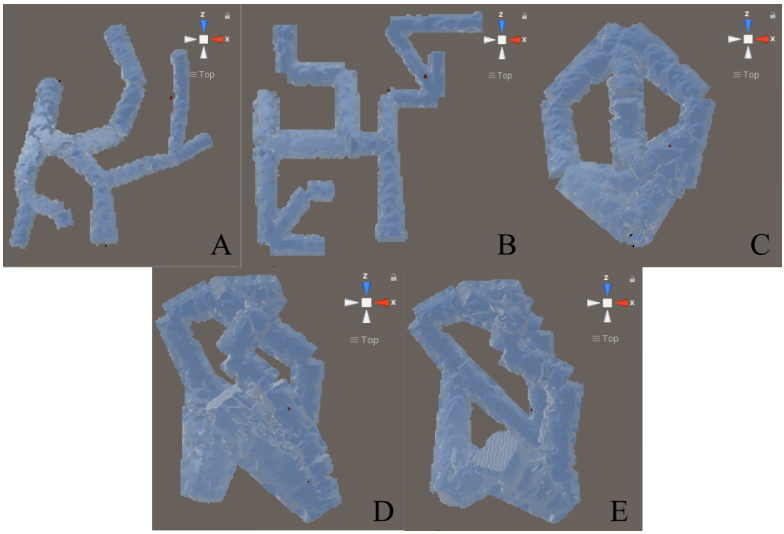
Terlihat pada tabel hasil pengujian bahwa mayoritas waktu yang dibutuhkan untuk menghasilkan model 3D semakin pendek ketika jumlah jalur gua semakin kecil.

4.4 Rekonstruksi Kelima Jenis Gua

Pengujian ini dilakukan dengan mencoba membentuk kelima jenis struktur jalur gua, yaitu *branchwork*, *network*, anastomotik, *ramiform*, dan *spongework*. Gua jenis *branchwork* dibuat dengan membuat *Node* pada posisi bawah, kemudian mulai membuat percabangan baru dari *Node* tersebut. Cabang baru dibuat pada beberapa *Node* selanjutnya. Setiap jalur yang dibuat tidak terhubung dengan *Node* lain yang akan menimbulkan *loop* pada struktur gua. Hal ini hanya muncul pada gua jenis anastomotik. Gua jenis *network* dibuat dengan mengatur ulang posisi seluruh *Node* dari gua jenis *branchwork* hingga terbentuk daerah terisi yang berbentuk menyerupai petak-petak. Rekonstruksi jenis *ramiform* dilakukan dengan mengatur jalur-jalur gua hingga ruang dari dua atau lebih jalur gua saling bersinggungan. Untuk rekonstruksi jenis *spongework* melalui tahap yang sama seperti rekonstruksi *ramiform*, namun jalur-jalur gua perlu diatur kembali sehingga terdapat beberapa daerah terisi pada ruang gua. Dari pengujian ini, kami mampu merekonstruksi



Gambar 4.8: Konfigurasi jalur-jalur gua dari sudut pandang atas. A: *Branchwork*, B: *Network*, C: Anastomotik, D: *Ramiform*, E: *Spongework*.

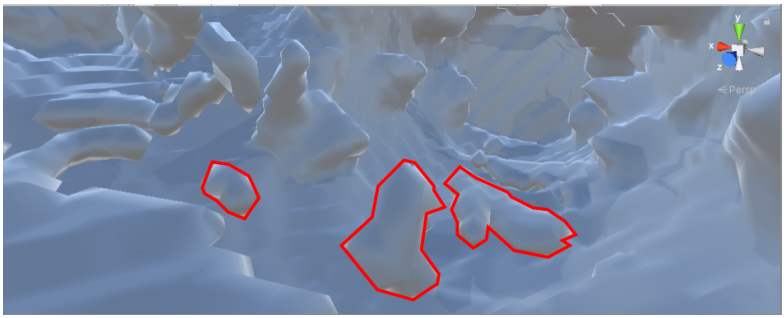


Gambar 4.9: Hasil rekonstruksi kelima jenis gua dari sudut pandang atas. A: *Branchwork*, B: *Network*, C: Anastomotik, D: *Ramiform*, E: *Spongework*.

kelima jenis gua. Namun begitu, terdapat beberapa kendala dan kekurangan dari metode yang kami tawarkan. Kendala yang pertama yaitu saat mengatur jalur-jalur gua diperlukan ketelitian sehingga dua atau lebih jalur gua saling bersinggungan untuk membuat gua jenis *ramiform* dan terdapat daerah terisi/ tembok yang tersisa untuk membuat gua jenis *spongework*. Pengaturan radius dan posisi *Node* secara manual dirasa terlalu sulit dan kurang mampu menciptakan percabangan yang lebih alami untuk merekonstruksi gua jenis *branchwork* dan anastomotik. Berbeda untuk rekonstruksi gua jenis *network* dirasa cukup mudah untuk mengatur posisi *Node* dan mendapatkan daerah terisi dalam daerah berpetak. Kekurangan dari metode ini adalah masih terdapat artifak melayang pada gua jenis *ramiform* dan *spongework*.

4.5 Survei Kepuasan Pengguna

Survei dilakukan dengan memberikan program kami dalam bentuk *Unity Package* dan panduan penggunaan kepada penguji.



Gambar 4.10: Daerah dengan garis merah adalah artifak melayang yang tidak diinginkan.

Penguji diminta untuk membuat minimal lima model 3D gua dengan nilai *parameter noise* dan voxel yang berbeda-beda, kemudian penguji mengisi sebuah kuesioner *feedback*. Penguji merupakan pengembang permainan video dengan posisi jabatan sebagai *3D artist* dan *game designer*. Kuesioner berisi pernyataan-pernyataan terkait program kami yang direspon dengan pernyataan setuju atau tidak setuju oleh penguji, serta bagian kritik dan saran dari penguji sebagai masukan untuk pengembangan penelitian selanjutnya. Hasil survei kami dari empat penguji ditampilkan pada tabel 4.9, dengan rincian dua penguji adalah mahasiswa yang berpengalaman sebagai *game designer* dan dua *3D artist*, satu yang masih berstatus mahasiswa dan satu karyawan pada sebuah perusahaan pengembang permainan video.

Dari survei ini, kami mengambil kesimpulan bahwa pengguna merasa puas dengan adanya program ini meskipun detil permukaan gua yang dihasilkan masih belum memuaskan pengguna. Mayoritas penguji merasa terbantu dengan adanya fitur pembuatan daerah jalan. Penguji juga mampu menghasilkan permukaan gua yang bervariasi sesuai keinginan mereka dengan mudah melalui perubahan parameter *noise* seperti *lacunarity*, *persistence*, dan *octave* serta parameter-parameter yang mengubah ukuran voxel dan resolusi *chunk*, namun penguji merasa biasa saja dengan parameter *noise seed*. Antar muka yang disediakan untuk mengubah struktur jalur gua dirasa cukup memudahkan penguji. Kami mendapatkan kritik

Tabel 4.8: Daftar pernyataan pada kuesioner.

No.	Pernyataan
1	”Saya merasa mudah dalam mengubah struktur jalur gua dengan mengubah posisi salah satu node pada tampilan ”Scene view””
2	”Saya memiliki kendali untuk mendapatkan detail permukaan gua yang bervariasi sesuai keinginan saya dengan mengubah resolusi chunk dan ukuran voxel”
3	”Saya memiliki kendali untuk mendapatkan permukaan gua yang bervariasi sesuai keinginan saya dengan mengubah nilai ”octave””
4	”Saya memiliki kendali untuk mendapatkan permukaan gua yang bervariasi sesuai keinginan saya dengan mengubah nilai ”persistence””
5	”Saya memiliki kendali untuk mendapatkan permukaan gua yang bervariasi sesuai keinginan saya dengan mengubah nilai ”lacunarity””
6	”Saya memiliki kendali untuk mendapatkan permukaan gua yang bervariasi sesuai keinginan saya dengan mengubah nilai ”noise seed””
7	”Saya merasa terbantu dengan opsi ”Simulate roadway creation” sehingga saya tidak perlu repot membuat permukaan halus yang digunakan sebagai daerah jalan”
8	”Saya merasa puas dengan detil permukaan gua yang dihasilkan dengan tools ini”
9	”Saya merasa puas dengan model 3D gua yang dihasilkan dengan tools ini”

Tabel 4.9: Hasil survei pengguna.

Pernyataan	Tanggapan				
	Sangat tidak setuju	Tidak setuju	Netral	Setuju	Sangat setuju
Pernyataan 1	0%	0%	25%	0%	75%
Pernyataan 2	0%	0%	25%	50%	25%
Pernyataan 3	0%	0%	25%	25%	50%
Pernyataan 4	0%	0%	25%	50%	25%
Pernyataan 5	0%	0%	25%	25%	50%
Pernyataan 6	0%	0%	50%	50%	0%
Pernyataan 7	0%	25%	0%	0%	75%
Pernyataan 8	0%	25%	0%	50%	25%
Pernyataan 9	0%	0%	25%	25%	50%

dan saran dari kedua penguji yang juga memiliki posisi sebagai *3D artist*. Penguji pertama mengkritik bahwa model 3D yang dihasilkan dari program kami dirasa sulit untuk dimodifikasi dari segi *UV-mapping*, penambahan detail pada gemotri model 3D gua, dan tidak adanya fitur *auto texture*. Penguji menyarankan agar antar muka program perlu diperbaiki sehingga pengguna dapat mengetahui bentuk ruang jalur gua terlihat seperti apa dari berbagai macam sudut pandang, namun tetap nyaman untuk dilihat.Penguji kedua memberi kritik yang sama terkait antar muka dengan saran berupa tambahan fitur perubahan radius *Node* langsung pada tampilan struktur jalur gua dan fitur untuk memilih lebih dari satu *Node*.

BAB 5
PENUTUP

5.1 Kesimpulan

Dari hasil pengujian yang sudah dilakukan dapat ditarik beberapa kesimpulan sebagai berikut:

1. Kami mampu menghasilkan lima jenis gua yang berbeda-beda menggunakan metode kami, namun terdapat artifak melayang pada gua jenis *ramiform*. Pembuatan gua jenis *branchwork* dan anastomotik dirasa masih sulit untuk menghasilkan percabangan dan ruang gua yang terlihat alami.
2. Perubahan parameter input *noise* memengaruhi data volume yang dihasilkan sehingga menghasilkan model 3D gua yang berbeda-beda. Perubahan *lacunarity* dapat memengaruhi tingkat kekasaran permukaan model 3D gua dan memunculkan artifak *spike* pada nilai yang lebih tinggi. Perubahan *persistence* dapat memengaruhi bentuk dari model 3D, namun tidak terlalu signifikan seperti pengaruh nilai *persistence*. Terakhir, perubahan *octave* selain memberikan dampak yang sama dengan pengaruh nilai *lacunarity*, waktu yang dibutuhkan untuk menghasilkan model 3D juga semakin lama. .
3. Perubahan resolusi voxel dapat memengaruhi kualitas dari model 3D yang dihasilkan beserta waktu yang dibutuhkan untuk menghasilkan model 3D tersebut. Semakin kecil dan semakin besar resolusi yang digunakan, semakin lama waktu yang dibutuhkan untuk menghasilkan model 3D gua.
4. Jumlah jalur gua yang diberikan dapat memengaruhi waktu yang dibutuhkan menghasilkan model 3D, sehingga disarankan untuk meminimalkan jumlah garis yang diperlukan, namun tetap menjaga struktur jalur gua.
5. Pengguna merasa puas dengan program kami dan model 3D yang dihasilkan, namun diperlukan perbaikan tampilan ruang jalur gua dan penambahan fitur *UV-mapping*, *auto texture*, dan fitur untuk memilih lebih dari satu *Node*. Pengguna merasa puas dengan adanya parameter-parameter seperti *lacunarity*, *persistence*, *octave*, resolusi *chunk* dan ukuran voxel

untuk menghasilkan permukaan gua yang berbeda-beda. Fitur pembuatan daerah jalan pada permukaan juga mengurangi pekerjaan pengguna untuk membuat daerah jalan.

5.2 Saran

Untuk pengembangan penelitian selanjutnya terdapat beberapa saran sebagai berikut :

1. Optimisasi dapat dilakukan dengan memeriksa apakah *chunk* mencakup satu atau lebih objek *Line*, sehingga menghemat waktu membuat data volume dan proses *marching cube* dengan tidak menyertakan *chunk* dengan seluruh vertex bernilai satu atau nol.
2. Melakukan tahap *post-processing* pada data volume yang telah dihasilkan sehingga mampu menghasilkan data volume yang lebih halus.
3. Algoritma-algoritma untuk menghasilkan jalur bercabang dapat diaplikasikan untuk meng-*generate* data jalur gua, seperti *L-system* dan konjektur Collatz.
4. *UV-mapping* otomatis dan *auto texture* dapat diaplikasikan sehingga model 3D gua yang dihasilkan, secara visual, dapat langsung diaplikasikan pada sebuah *video game* dan meringankan beban 3D *artist* dalam mengubah visual model 3D gua.

DAFTAR PUSTAKA

[1] S. Lague, "Coding adventure: Marching cubes," March 2019. (Dikutip pada halaman , 35).

[2] D. Carli, F. Bevilacqua, C. Pozzer, and M. d'Ornellas, "A survey of procedural content generation techniques suitable to game development," pp. 26–35, November 2011. (Dikutip pada halaman 1).

[3] C. Solarski, "The aesthetics of game art and game design," January 2013. (Dikutip pada halaman 1).

[4] N. Simon, "What we talk about when we talk about game aesthetics," in DiGRA '09 - Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory, Brunel University, September 2009. (Dikutip pada halaman 1).

[5] J. Cui, Y.-W. Chow, and M. Zhang, "Procedural generation of 3d cave models with stalactites and stalagmites," International Journal of Computer Science and Network Security, vol. 11, August 2011. (Dikutip pada halaman 2, 13).

[6] J. Cui, Y.-W. Chow, and M. Zhang, "A voxel-based octree construction approach for procedural cave generation," International Journal of Computer Science and Network Security, vol. 11, January 2011. (Dikutip pada halaman 2, 13).

[7] B. Mark, T. Berechet, T. Mahlmann, and J. Togelius, "Procedural generation of 3d caves for games on the gpu," June 2015. (Dikutip pada halaman 3, 13).

[8] A. Palmer, "Palmer an.. origin and morphology of limestone caves. geol soc am bull 103: 1-21," Geological Society of America Bulletin - GEOL SOC AMER BULL, vol. 103, pp. 1–21, 01 1991. (Dikutip pada halaman 7).

[9] S. Libeskind, Euclidean and Transformational Geometry: A Deductive Inquiry. Jones & Bartlett Learning, 2008. (Dikutip pada halaman 8, 25).

[10] P. Bourke, “Pipes for rendering engines,” 1999. (Dikutip pada halaman 8, 17).

[11] K. Perlin, “Noise hardware,” ACM SIGGRAPH Computer Graphics, 2001. (Dikutip pada halaman 10).

[12] S. Gustavson, “Simplex noise demystified,” March 2005. (Dikutip pada halaman 10).

[13] W. Lorensen and H. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” ACM SIGGRAPH Computer Graphics, vol. 21, pp. 163–, August 1987. (Dikutip pada halaman 10).

[14] P. Bourke, “Polygonising a scalar field,” 1994. (Dikutip pada halaman 11, 35).

[15] Microsoft, “Compute shader overview,” May 2018. (Dikutip pada halaman 11).

[16] U. Technologies, “Compute shaders,” May 2017. (Dikutip pada halaman 11).

[17] Microsoft, “numthreads,” May 2018. (Dikutip pada halaman 11).

[18] W. Wijaya and A. Rahman, “Analisis perbandingan perlin noise dan simplex noise untuk penciptaan permukaan daratan pada pembuatan game,” Konferensi Nasional Sistem Informasi (KNSI) 2018, 2018. (Dikutip pada halaman 27).

[19] S. Setarehdan and S. Singh, Advanced Algorithmic Approaches to Medical Image Segmentation: State-of-the-Art Applications in Cardiology, Neurology, Mammography and Pathology. Advances in Computer Vision and Pattern Recognition, Springer London, 2012. (Dikutip pada halaman 32).

[20] J. Suri and S. Laxminarayan, Angiography and Plaque Imaging: Advanced Segmentation Techniques. Biomedical Engineering, CRC Press, 2003. (Dikutip pada halaman 32).

BIOGRAFI PENULIS

Khalil Satyadama lahir di Banyuwangi, Jawa Timur pada tanggal 16 November 1998. Merupakan anak pertama dari empat bersaudara. Penulis lulus dari SMA Negeri 4 Semarang pada tahun 2016, kemudian melanjutkan studi ke jenjang strata satu di Departemen Teknik Komputer, Fakultas Teknologi Elektro dan Informatika Cerdas. Ketertarikan penulis dengan permainan video beserta proses pengembangannya dan hobi menggambar membawa penulis untuk memelajari *3D modelling* secara otodidak dan membuat desain serta *prototype* permainan video. Penulis juga merupakan asisten laboratorium telematika B201 serta ketua *development group* prodi *game technology* laboratorium B201 periode 2017-2018. Di tahun keempat studi, penulis menjadi *2D artist* dengan nama pena "*kopi SACHet*". Karya yang telah dibuat adalah karakter original berbentuk cangkir yang biasa digunakan untuk menikmati kopi dan dua permainan video, masing-masing adalah permainan dengan *genre top-down shooter* dan *shoot-em-ups*.

Halaman ini sengaja dikosongkan