BAB 3

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Pelaksanaan magang di Nudge Nudge Games GmbH difokuskan dalam bagian *graphics programming*, dengan Bapak Rafael Wawer selaku direktur sebagai pembimbing. Bapak Rafael Wawer berperan dalam memberikan tugas yang perlu dikerjakan dan memeriksa tugas yang telah diselesaikan.

3.2 Tugas Yang Dilakukan

Dalam pelaksanaan magang, beberapa tugas yang diberikan adalah sebagai berikut:

- Menyelesaikan prototipe grafik yang telah dikerjakan sebelum kerja magang.
- Mengkonversi *shader* untuk mendukung *Universal Render Pipeline* (URP)
- Mengimplementasi beberapa fitur grafik ke dalam Nimoyd.
- Memperbaiki fitur *texture packing* untuk mendukung *texture* dengan resolusi lebih tinggi.
- Memperdalam *voxel meshing* untuk menghasilkan model *voxel* dengan tepi bulat.
- Memperbaiki normal mapping untuk *voxel*.

3.3 Uraian Pelaksanaan Kerja Magang

3.3.1 Proses Pelaksanaan

Kerja magang dilaksanakan dalam jangka waktu 12 minggu (atau 3 bulan) dengan *timeline* kerja sebagai berikut:

Tabel 3.1 Timeline kerja magang

Nama Kegiatan	Minggu											
	1	2	3	4	5	6	7	8	9	10	11	12
Menyelesaikan prototipe grafik												
Konversi shader ke URP												
Implementasi fitur grafik												
Memperbaiki texture packing												
Menambah tepi bulat ke voxel meshing												
Normal mapping untuk voxel												

Sebuah prototipe grafik yang dibangun menggunakan *game engine* Unity diselesaikan dalam tiga minggu pertama kerja magang. Setelah prototipe grafik selesai dibangun, dilakukan konversi *shader* dalam Nimoyd dari *Built-in Render Pipeline* ke *Universal Render Pipeline* (URP), mulai dari minggu ketiga sampai minggu ketujuh. Implementasi berbagai fitur grafik dilakukan seiring konversi *shader* dalam minggu keempat sampai minggu keenam.

Perbaikan pada fitur *texture packing* dikerjakan pada minggu kedelapan dan minggu kesembilan, implementasi tepi bulat pada *voxel meshing* dikerjakan pada minggu kesembilan sampai minggu terakhir, dan perbaikan *normal mapping* untuk *voxel* dikerjakan pada dua minggu terakhir.

3.3.2 Perancangan

A. Konversi Shader ke URP

Konversi *shader* dilakukan karena efek *godrays* dari prototipe grafik hanya bisa dilakukan dalam URP dan efek tersebut akan diimplementasikan dalam Nimoyd. *Shader* yang dikonversi dijabarkan dalam Tabel 3.2.

Tabel 3.2 Konversi *shader Built-in* ke URP

Jenis *Shader*	Deskripsi
Voxel Basic	Digunakan pada *voxel* yang menggunakan 1 tekstur untuk seluruh permukaan
Voxel Top Side	Digunakan pada *voxel* yang menggunakan 2 tekstur untuk *triplanar mapping*
Voxel Billboard	Digunakan pada *voxel* yang menggunakan efek *billboard*
Voxel Water	Digunakan pada *voxel* air
Model Basic	Digunakan pada model dengan *texture* non-transparan
Model Basic Cutout	Digunakan pada model dengan *texture* yang memiliki bagian transparan
Model Billboard	Digunakan pada model yang menggunakan efek *billboard*
Model Billboard Animated	Digunakan pada model *billboard* dengan animasi *spritesheet*

Tabel 3.3 Konversi *shader Built-in* ke URP (lanjutan)

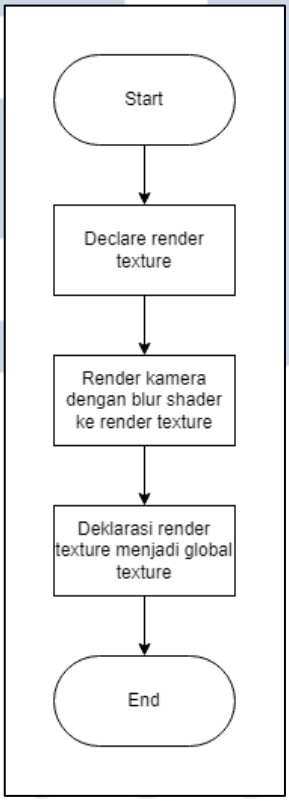
Jenis *Shader*	Deskripsi
Model Billboard Bend	Digunakan pada model yang dapat membengkok
Model Billboard Merged	Digunakan pada model yang menggunakan *pivot caching*
Model Sprite	Digunakan pada partikel
Model Sprite Billboard	Digunakan pada karakter (pemain maupun NPC)
UI Overlay	Digunakan pada UI dalam *world space*

B. Implementasi Fitur Grafik

Terdapat beberapa fitur grafik yang diimplementasikan dalam Nimoyd. Fitur grafik yang diambil dari prototipe adalah *pixel filter*, *background blur*, *godrays*, dan *screen space particle*. Fitur grafik lainnya yang diimplementasi adalah *mesh light*, *water foam*, dan *water fog*.

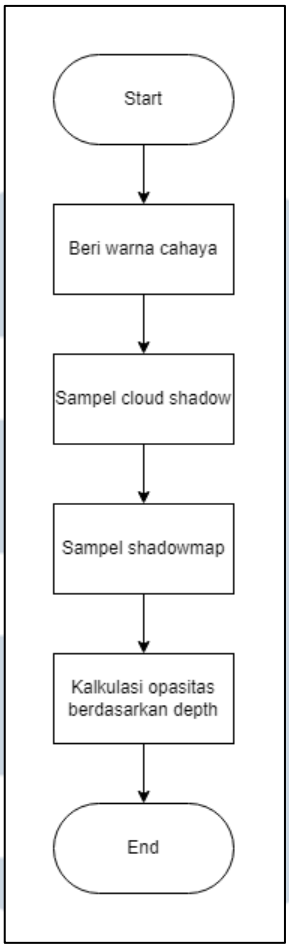
Pixel filter merupakan fitur grafik yang membuat tampilan terlihat seperti memiliki resolusi rendah. Fitur ini diimplementasikan menggunakan dua kamera yang ditempatkan pada posisi yang sama. Kamera pertama merender seluruh objek kecuali *interface* ke sebuah *texture dengan resolusi rendah*. *Texture* yang menyimpan hasil kamera pertama digunakan oleh sebuah objek gambar UI. Kamera kedua hanya merender objek *interface. Pixel filter* awalnya digunakan untuk menguatkan gaya visual *pixel art*, namun pada akhirnya dihapus karena akan dilakukan perubahan gaya visual dari *pixel art* ke 3 dimensi kartun/non-realistik.

Background blur adalah sebuah fitur grafik yang memburamkan tampilan dari *game* kecuali pada *interface*. Fitur ini diimplementasikan menggunakan *render feature*, yang memungkinkan penggunaan instruksi grafik kapan saja dalam proses *rendering*. *Render feature* dari *background blur* menggunakan kamera utama untuk *render* ke sebuah *texture* dengan resolusi rendah menggunakan *bicubic filtering*. *Texture* tersebut dideklarasikan sebagai *global shader texture* yang kemudian digunakan oleh *shader* dari objek gambar UI.



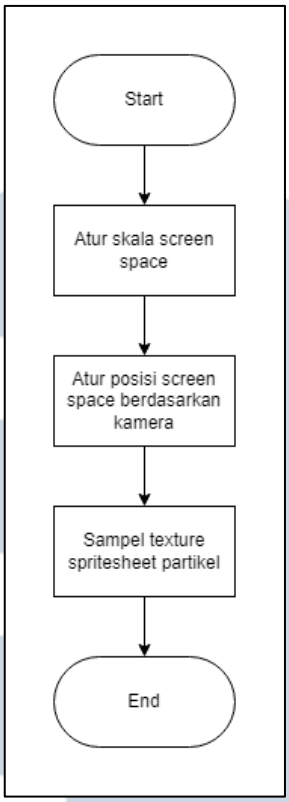
Gambar 3.1 *Flowchart* dari algoritma *background blur*

Godrays merupakan fitur grafik yang menampilkan sinar matahari. Fitur ini diimplementasikan menggunakan sejumlah bidang datar semitransparan yang menghadap ke kamera utama dan *shader* yang menyampel bayangan dan memudarkan opasitas melalui *depth texture*.



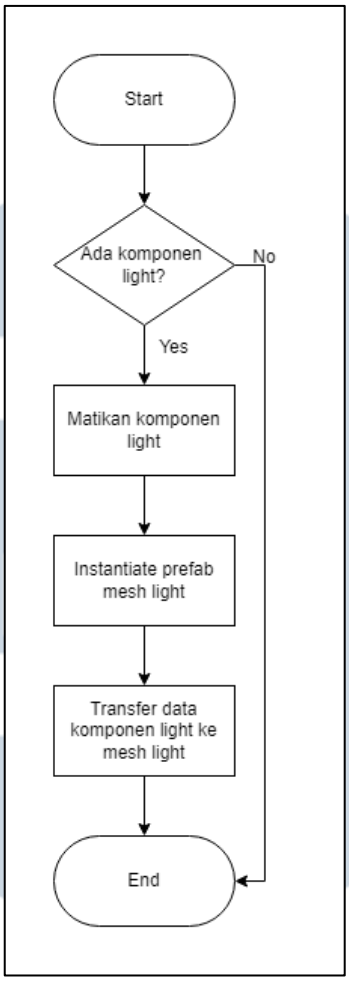
Gambar 3.2 *Flowchart* dari *shader godrays*

Screen space particle adalah fitur grafik yang menampilkan sekumpulan partikel dalam layar penuh menggunakan spritesheet. Fitur ini hanya dapat digunakan apabila kamera menggunakan mode ortografi atau memiliki sudut pandang yang sangat rendah. Posisi dari partikel diatur berdasarkan gerakan dan perputaran kamera yang dikalkulasi dalam *script* kamera dan dipindahkan ke shader menggunakan *global shader variable*.



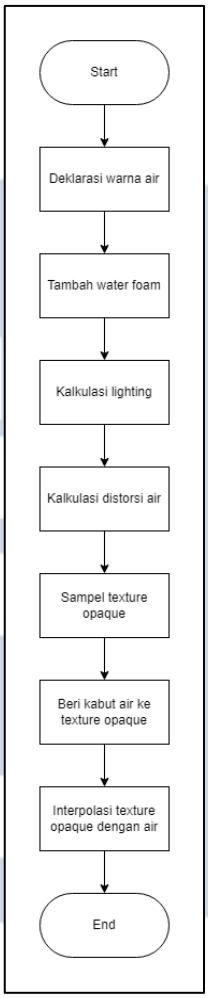
Gambar 3.3 *Flowchart* dari *shader screen space particle*

Mesh light merupakan fitur grafik yang digunakan untuk memberikan cahaya dari partikel. Fitur ini digunakan adanya batasan dari URP dimana satu objek hanya dapat disinari oleh 8 objek *light.* Nimoyd menggunakan partikel bercahaya dengan jumlah banyak dalam kondisi tertentu, dan menggunakan komponen *light* dapat menyebabkan cahaya hanya terlihat sebagian. Mesh light diimplementasikan menggunakan sebuah *script* yang mematikan komponen *light* yang kemudian digantikan oleh objek mesh light berupa bola dengan material khusus. Properti dari komponen *light* dipindahkan ke material, kecuali properti jarak yang digunakan sebagai skala bola.



Gambar 3.4 *Flowchart* dari *script* konvertor *mesh light*

Shader air dalam Nimoyd ditingkatkan dengan membuat ulang fitur *water foam* dan menambahkan *water fog*. *Water foam* dibentuk menggunakan gradien yang arahnya ditentukan dari data uv dalam *mesh*, dibantu dengan *texture* tambahan. *Water fog* diimplementasikan melalui interpolasi ke suatu warna berdasarkan jarak dari permukaan dalam air ke permukaan air itu sendiri.

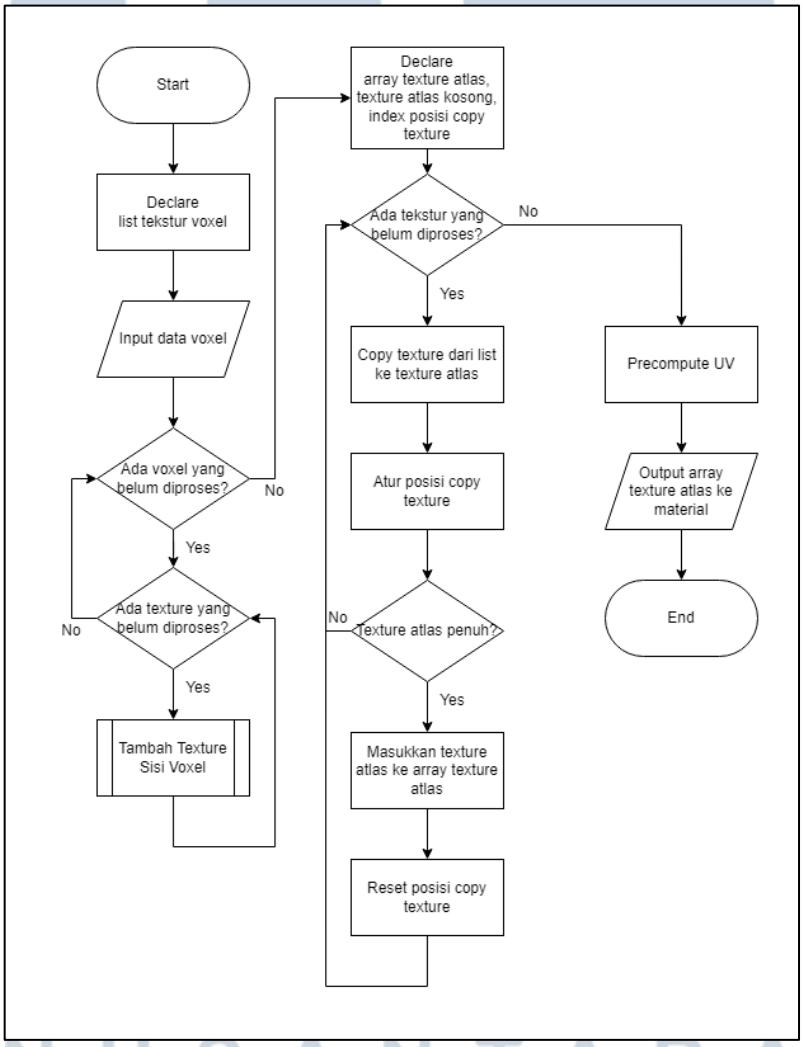


Gambar 3.5 Flowchart dari *shader* air

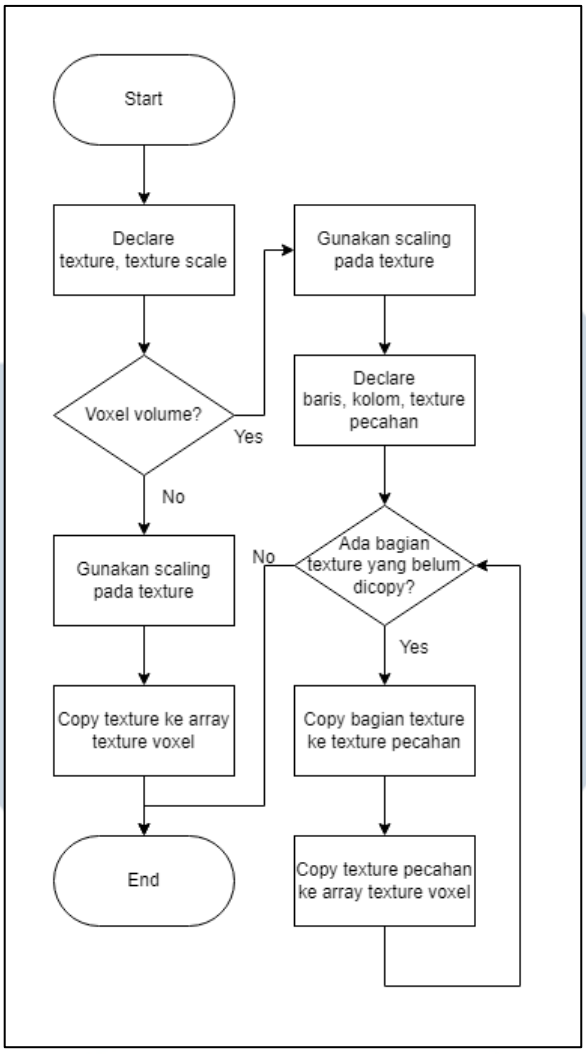
C. Memperbaiki Texture Packing

Texture packing adalah sebuah proses penyusunan banyak *texture* menjadi sebuah *texture* besar yang dikenal sebagai *texture atlas*. Dalam Nimoyd, terdapat *voxel* dengan ukuran yang lebih besar dari 1x1x1 *unit* yang dikenal dalam projek sebagai *volume voxel*. *Texture* dari *volume voxel* dipecah menjadi beberapa bagian dengan ukuran 1x1 *unit* sebelum disusun, dimana 1 unit merepresentasikan panjang dan lebar *texture voxel* yang ditentukan yaitu 16 *pixel*. *Texture* yang dimiliki setiap *voxel* disusun menjadi beberapa *texture atlas*.

Proses pemecahan *texture volume voxel* tidak berfungsi dengan baik saat ukuran *texture* diperbesar dari 16 *pixel* menjadi 64 *pixel*, menyebabkan jumlah *texture atlas* yang dibuat lebih sedikit dari yang seharusnya. Kegagalan pada pemecahan *texture volume voxel* disebabkan oleh tidak adanya pengaturan ukuran *texture* sebelum *texture* tersebut dipecah. Fitur pengaturan ukuran tekstur telah diimplementasikan pada *texture* yang dimiliki voxel biasa, fitur yang sama dapat digunakan pada *texture volume* untuk mengatasi masalah yang telah dijelaskan.



Gambar 3.6 *Flowchart* dari algoritma *texture packing*

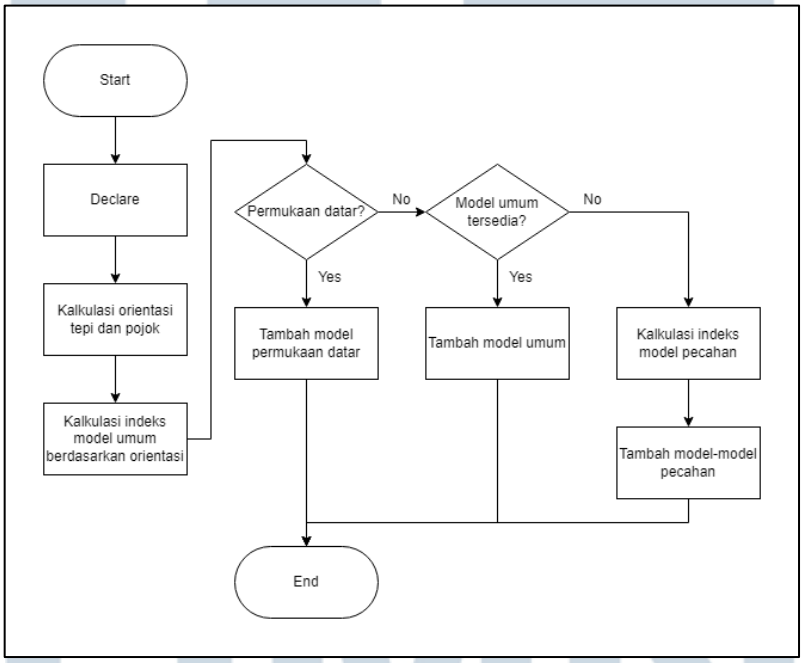


Gambar 3.7 *Flowchart* dari metode penambahan *texture* sisi *voxel*

D. Menambah Tepi Bulat ke Voxel Meshing

Dunia *voxel* umumnya dibagi menjadi sejumlah wilayah dengan ukuran yang sama, dikenal sebagai *chunk*. Setiap *chunk* dimodelkan sebagai sekumpulan kubus, dimana permukaan pada kubus hanya dibangun apabila tidak ada *voxel* lain di depan permukaan tersebut. Sebuah tugas diberikan untuk menambah tepi bulat/halus pada model voxel. Solusi yang digunakan adalah membagi permukaan voxel menjadi 4 bagian dan menggunakan sejumlah model yang merepresentasikan seperempat permukaan voxel dengan tepi bulat. Model yang digunakan ditentukan berdasarkan keberadaan voxel di sekitar permukaan.

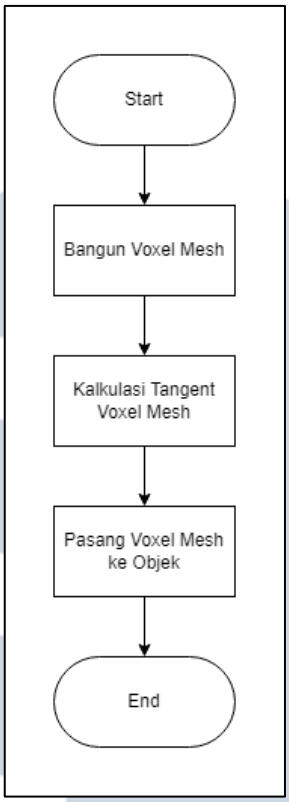
Melakukan pembagian pada seluruh permukaan voxel akan menghasilkan model dengan jumlah vertex dan triangle yang sangat tinggi, oleh karena itu pembagian permukaan tidak dilakukan dalam kondisi tertentu. Salah satu kondisi tersebut adalah bentuk yang dihasilkan sangat umum, sehingga menggunakan model permukaan utuh. Kondisi yang lain adalah voxel di sekitar permukaan menghasilkan permukaan datar, sehingga menggunakan pembangunan permukaan datar yang telah disediakan.



Gambar 3.8 *Flowchart* dari algoritma pembangunan permukaan voxel

E. Memperbaiki Normal Mapping untuk Voxel

Normal mapping merupakan sebuah fitur grafik yang membuat suatu permukaan menjadi lebih timbul dengan menggunakan *texture* yang dikenal sebagai *normal map*. Fitur ini membutuhkan data *tangent* dalam mesh, namun *voxel meshing* menghasilkan mesh yang tidak memiliki *tangent*. Unity menyediakan sebuah fungsi untuk menghitung *tangent* pada mesh, yang akan diimplementasikan berdasarkan *flowchart* dalam Gambar 3.9.



Gambar 3.9 *Flowchart* pengaplikasian kalkulasi *tangent* dalam *voxel meshing*

Normal map disediakan ke *voxel* melalui texture packing, namun texture atlas tidak terisi sepenuhnya saat fitur normal mapping yang telah disediakan diaktifkan. Hal ini disebabkan oleh iterasi *texture* warna dan *texture normal* yang bergantian sehingga membentuk wilayah kosong dalam *texture atlas*, dan normal map polos yang dideklarasikan untuk *volume voxel* memiliki ukuran 1x1 unit sehingga tidak dapat dibagi dan menghasilkan *error*.

3.3.3 Hasil Implementasi

A. Implementasi Fitur Grafik

Hasil implementasi fitur grafik dari prototipe seperti pixel filter, background blur, godrays, dan screen space particle ditampilkan pada Gambar 3.10, Gambar 3.11, Gambar 3.12, dan Gambar 3.13.

Gambar 3.10 Hasil dari implementasi *pixel filter*

Gambar 3.11 Hasil dari implementasi *background blur*

Gambar 3.12 Hasil dari implementasi godrays

Gambar 3.13 Hasil dari implementasi *screen space particle*

Hasil dari implementasi mesh light ditampilkan pada sisi kanan dari gambar 3.14. Penggunaan komponen light yang melebihi batas ditampilkan pada sisi kiri. Water fog dan water foam ditampilkan pada gambar 3.15 dan gambar 3.16.

Gambar 3.14 Hasil dari implementasi *mesh light*

Gambar 3.15 Hasil dari implementasi *water fog*

Gambar 3.16 Hasil dari implementasi *water foam*

B. Memperbaiki Texture Packing

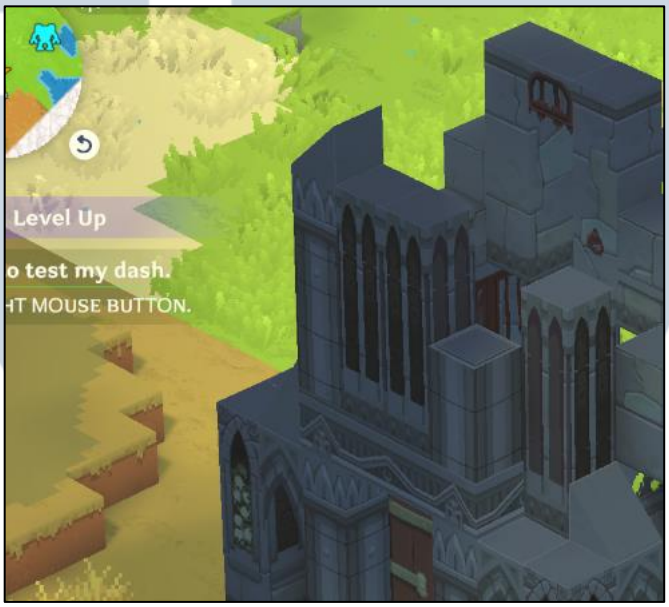
Kumpulan *texture atlas* yang tidak lengkap ditampilkan pada Gambar 3.17. *Texture atlas* pada gambar tersebut hanya menampilkan *texture* yang dimiliki *voxel* selain *voxel volume*. Gambar 3.18 menampilkan kumpulan *texture atlas* yang lengkap, dimana seluruh *texture* dari *voxel volume* berhasil dimasukkan ke dalam *texture atlas*.

Gambar 3.17 Sejumlah *texture atlas* sebelum perbaikan *texture packing*

Gambar 3.18 Sejumlah *texture atlas* setelah perbaikan *texture packing*

C. Menambah Tepi Bulat ke Voxel Meshing

Hasil dari voxel meshing yang mendasar ditampilkan pada Gambar 3.19. Hasil dari voxel meshing dengan tepi bulat ditampilkan pada Gambar 3.20. Bagian yang tidak memiliki tepi bulat merupakan model terpisah yang tidak dibangun oleh voxel meshing.

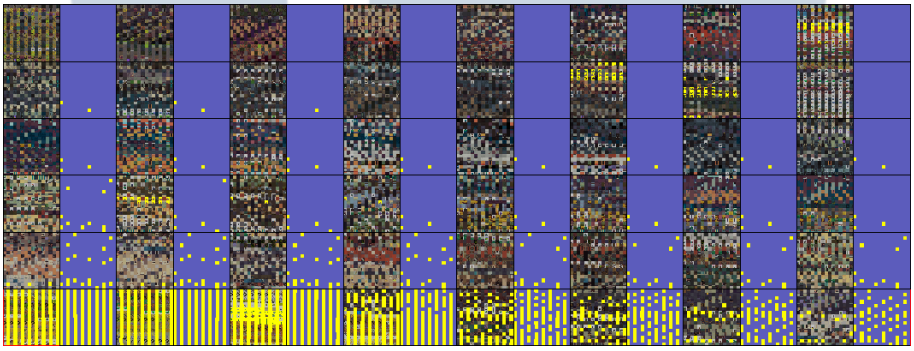


Gambar 3.19 Voxel dengan tepi normal

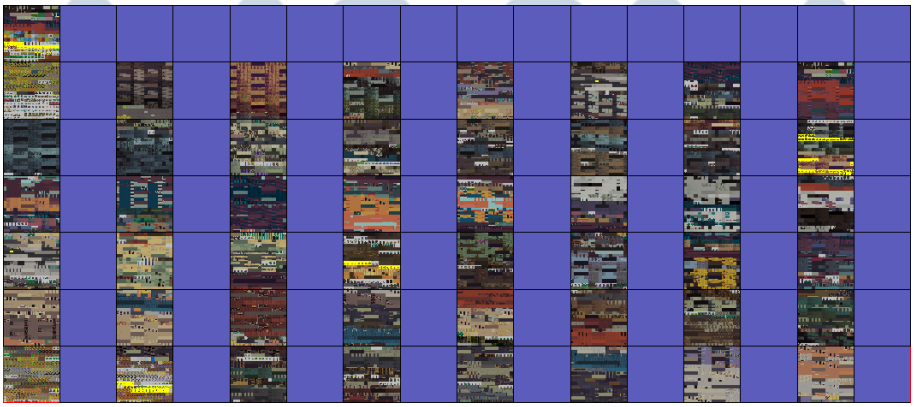
Gambar 3.20 Voxel dengan tepi bulat

D. Memperbaiki Normal Mapping untuk Voxel

Kumpulan *texture atlas* warna dan *normal* sebelum perbaikan ditampilkan pada Gambar 3.21, wilayah transparan atau kosong ditandakan dengan warna kuning. Gambar 3.22 menampilkan kumpulan *texture atlas* warna dan *normal* setelah perbaikan, dimana wilayah kosong berhasil diperbaiki dan warna kuning hanya menandakan wilayah transparan. Hasil dari *normal mapping* ditampilkan pada Gambar 3.23. *Normal map* yang digunakan pada gambar tersebut merupakan *texture* tunggal di luar kumpulan *texture atlas*.



Gambar 3.21 Sejumlah *texture atlas* warna dan *normal* sebelum perbaikan



Gambar 3.22 Sejumlah *texture atlas* warna dan *normal* setelah perbaikan

Gambar 3.23 Implementasi *normal mapping* pada voxel

3.3.3 Kendala yang Ditemukan

Dalam pelaksanaan kerja magang, terdapat beberapa kendala yang ditemukan sebagai berikut:

- Dokumentasi yang disediakan Unity mengenai URP sangat sedikit di luar *shader graph*.
- Metode pertama tepi bulat pada voxel meshing menghasilkan triangle count yang terlalu tinggi.

3.3.4 Solusi Atas Kendala yang Ditemukan

Solusi yang telah ditemukan untuk mengatasi kendala yang telah dijelaskan adalah sebagai berikut:

- Menggunakan dokumentasi dan *template* kode yang disediakan oleh komunitas.
- Menggunakan model dengan jumlah *vertex* dan *triangle* yang lebih rendah.