FINAL PROJECT – TE 141599

DATA LIBRARY AND VISUALIZATION OF 3D OBJECTS FOR DIGITAL MUSEUM CURATOR

Lintang Anugrah
NRP 2210100047

Adviser
Dr. Eko Mulyanto Yuniarno, ST., MT.
Ahmad Zaini, ST., M.Sc.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Industrial Technology
Sepuluh Nopember Institute of Technology
Surabaya 2017

Halaman ini sengaja dikosongkan

TUGAS AKHIR – TE 141599

PUSTAKA DATA DAN VISUALISASI CITRA 3D UNTUK KURATOR MUSEUM DIGITAL

Lintang Anugrah
NRP 2210100047

Dosen Pembimbing
Dr. Eko Mulyanto Yuniarno, ST., MT.
Ahmad Zaini, ST., M.Sc.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh November
Surabaya 2017

Halaman ini sengaja dikosongkan

PERNYATAAN KEASLIAN

TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**Pustaka Data dan Visualisasi Citra 3D untuk Kurator Museum Digital**" 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, 02 Agustus 2017

Lintang Anugrah
NRP. 2210100047

Halaman ini sengaja dikosongkan

PUSTAKA DATA DAN VISUALISASI CITRA 3D UNTUK KURATOR MUSEUM DIGITAL

TUGAS AKHIR

Diajukan Guna Memenuhi Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Teknik Komputer dan Telematika
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui:

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Eko Mulyanto Yuniarno, ST., MT.	**Ahmad Zaini, ST., M.Sc.**
NIP 196806011995121009	NIP 197504192002121003

SURABAYA
JULI, 2017

Halaman ini sengaja dikosongkan

ABSTRAK

Implementasi pustaka data museum untuk objek pindaian 3D koleksi biasanya akan menggunakan penampil *embedded viewer* dan *file* objek 3D tersendiri yang terpisah dalam folder tersendiri dari *database website*, dimana antarmuka situs akan memanggil *database* untuk mencari *listing* dari *file* di dalam *database*, yang akan dipanggil dan akan di-*forward* ke antarmuka untuk ditampilkan menggunakan *embedded viewer*. Prosedur ini dapat memiliki komplikasi tertentu, semisal besarnya wilayah penyimpanan yang dialokasikan untuk *file* objek 3D dan kemungkinan kesalahan dalam pendataan *file*. Pada tugas akhir ini akan diajukan implementasi penyimpanan *file* berformat OBJ dan MTL dalam *database* secara langsung yang dapat diakses oleh *viewer*.

Kata kunci :Visualisasi, *Database*, *Website*

Halaman ini sengaja dikosongkan

ABSTRACT

Data library implementations in museums for 3D visualization of scanned collections usually will use an embedded viewer, which calls a separate object file in its own folder separate from the website database, where the site interface will call the database to look for the file listing and directory inside before being forwarded to the viewer to be displayed to the user. This approach may have its own drawbacks, such as size of storage occupied by the separate object files and problems in registrating each file listing. This project aims to implement a storage system of object files formatted as .OBJ and .MTL files directly inside the database to be called by the viewer.

Keyword : Visualization, Database, Website

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji dan Syukur kehadirat Tuhan Yang Maha Esa atas segala limpahan berkah serta rahmat-Nya maka penulis dapat menyelesaikan Tugas Akhir penelitian ini dengan mudah dan selesai tepat waktu. Penelitian ini disusun dalam rangka pemenuhan bidang riset di Jurusan Teknik Elektro ITS, Bidang Studi Teknik Komputer dan Telematika, serta digunakan sebagai persyaratan menyelesaikan pendidikan S-1. Terimakasih penulis haturkan kepada pihak-pihak yang terlibat dalam proses penelitian ini maka dapat terselesaikan tidak lepas dari bantuan berbagai pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Keluarga, Ibu, Bapak dan Saudara, serta kerabat yang telah memberikan dorongan spiritual dan material
2. Bapak Dr. Tri Arief Sardjono, S.T., M.T. selaku Wakil Dekan Fakultas Teknologi Industri, Institut Teknologi Sepuluh Nopember.
3. Bapak Dr. Eng. Ardyono Priyadi, ST., M.Eng. selaku Ketua Jurusan Teknik Elektro-ITS.
4. Bapak Dr. Eko Mulyanto Yuniarno, ST., MT. dan Bapak Ahmad Zaini, ST, M.Sc. selaku dosen pembimbing yang baik dan perhatian.
5. Bapak-ibu dosen pengajar Bidang Studi Teknik Komputer dan Telematika, atas pengajaran, bimbingan, serta perhatian yang diberikan kepada penulis selama ini.
6. Seluruh teman-teman angkatan e-50 serta teman-teman B201 Laboratorium Bidang Studi Teknik Komputer dan Telematika.

Kesempurnaan hanya milik Tuhan, untuk itu penulis memohon segenap kritik dan saran yang membangun serta menghatur maaf atas segala kekurangan yang ada dalam penulisan buku ini. Semoga penelitian ini dapat memberikan manfaat bagi kita semua. Amin.

Surabaya, 02 Agustus 2017

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

LIST OF CONTENTS

DAFTAR GAMBAR

LIST OF PICTURES

DAFTAR TABEL

Halaman ini sengaja dikosongkan

LIST OF TABLES

Halaman ini sengaja dikosongkan

DAFTAR KODE

Halaman ini sengaja dikosongkan

LIST OF CODES

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

1.1. Latar Belakang

Suatu museum biasanya akan memiliki banyak koleksi benda bersejarah yang tidak terhitung nilainya. Benda bersejarah biasanya akan dikelompokkan dalam berbagai grup sesuai dengan sejarah dari benda tersebut dan relasinya dengan benda-benda lain yang terdapat dalam koleksi museum tersebut. Pengelompokkan dan pengurusan benda-benda koleksi akan menjadi tanggung jawab dari kurator museum.

Seiring dengan perkembangan zaman, daya dukung pendataan akan semakin menguat. Suatu museum dulunya mungkin hanya memiliki daftar koleksi dalam bentuk tertulis, namun sekarang kebanyakan museum di luar negeri telah memiliki daftar berbentuk digital untuk membantu mempercepat proses katalok koleksi benda bersejarah. Selain itu, muncul juga upaya untuk melestarikan benda bersejarah yang sudah tua atau mulai rapuh, salah satunya adalah dengan pencitraan digital benda nyata menjadi model digital 3D. Model 3D ini akan mendukung dalam penampilan benda-benda bersejarah, dimana kurator dapat menampilkan objek tanpa perlu mengeluarkan benda aslinya ketika benda sedang dalam restorasi, atau ketika perlu menampilkan benda di tempat dimana tidak bisa membawa benda tersebut, atau dalam restorasi benda dimana model dapat diubah untuk menjadi pemandu restorasi.

Skema ini juga dapat menjadi keuntungan bagi masyarakat luas dimana seseorang dapat melihat benda-benda bersejarah dalam bentuk citra 3D, atau dapat melihat katalog *online* sebagai bentuk kurator atau pemandu digital untuk menentukan koleksi mana yang ingin dilihat olehnya. Selain itu, museum juga dapat merilis koleksi digital untuk publik sebagai pengayaan masyarakat. Dalam skema keseluruhan ini, inti yang paling utama adalah bagaimana menyusun suatu sistem yang dapat mendukung kumpulan koleksi digital citra 3D benda bersejarah beserta dengan katalog identifikasi setiap objek tersebut. Selain itu, harus terdapat

juga sistem visualisasi citra 3D dari koleksi digital tersebut agar dapat dinikmati oleh masyarakat.

1.2. Permasalahan

Museum di Indonesia kebanyakan belum memiliki pustaka data yang dapat mendukung penyimpanan data model 3D hasil digitalisasi beserta dengan katalog penyimpanan koleksi. Selain itu, museum yang telah memiliki pustaka data juga belum memiliki penampil model untuk membantu dalam penggunaan model dalam rangka aktivitas seperti restorasi koleksi dan lain-lain.

1.3. Tujuan

Penelitian ini bertujuan untuk menerapkan suatu implementasi penyatuan penyimpanan berkas citra 3D dalam bentuk format kolektif *Wavefront OBJ* dengan *database* indeks dari berkas tersebut, serta penampil yang dapat membaca *database* bersatu dan mengubahnya menjadi penampilan citra yang diinginkan oleh pengguna. Indeksasi dan penggunaan fitur pencarian juga diterapkan agar dapat membuat pengguna mudah menggunakan *database* tersebut. Implementasi ini diharapkan dapat membantu pengguna mencari dan menampilkan objek 3D yang diinginkan, beserta dengan info data yang berhubungan dengan cepat dan dengan susunan yang lebih terstruktur.

1.4. Batasan Masalah

Dalam pengerjaan tugas akhir ini, diberikan beberapa batasan masalah. Diantaranya adalah sebagai berikut:

1. Implementasi menggunakan sumber berkas objek 3D yang telah disediakan oleh dosen pembimbing sebagai bahan pengisian sistem dan sebagai bahan pengujian sistem.
2. Mengikuti poin 1, implementasi tidak menggunakan berkas model yang memiliki tampak dalam.
3. Implementasi database menggunakan MySQL
4. Implementasi 3D *object viewer* menggunakan WebGL dan Three.js

1.5. Sistematika Penulisan

Laporan penelitian tugas akhir ini tersusun secara terstruktur sehingga lebih mudah dipahami dan dipelajari oleh pembaca maupun pihak yang hendak 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, tujuan penelitian, metodologi penelitian dan sistematika laporan.

2. Bab II Dasar Teori
   Pada bab ini berisi tentang uraian secara sistematis teori-teori yang berhubungan dengan permasalahan yang dibahas pada penelitian ini. Teori-teori ini digunakan sebagai dasar dalam penelitian.

3. Bab III Perancangan Sistem dan Implementasi
   Bab ini berisi tentang penjelasan-penjelasan terkait sistem yang akan dibuat. Guna mendukung itu digunakanlah blok diagram agar sistem yang akan dibuat dapat terlihat dan mudah dibaca untuk diimplentasikan pada pembuatan perangka.

4. Bab IV Pengujian dan Analisis
   Bab ini menjelaskan tentang pengujian yang dilakukan terhadap sistem dalam penelitian ini dan menganalisa sistem. Spesifikasi perangkat keras dan perangkat lunak yang digunakan juga disebutkan dalam bab ini. Tujuannya adalah sebagai variable kontrol dari pengujian yang dilakukan.

5. Bab V Penutup
   Bab ini merupakan penutup yang berisi kesimpulan yang diambil dari penelitian dan pengujian yang telah dilakukan. Saran dan kritik yang membangun untuk pengembangkan lebih lanjut juga dituliskan pada bab ini.

1.6. Relevansi

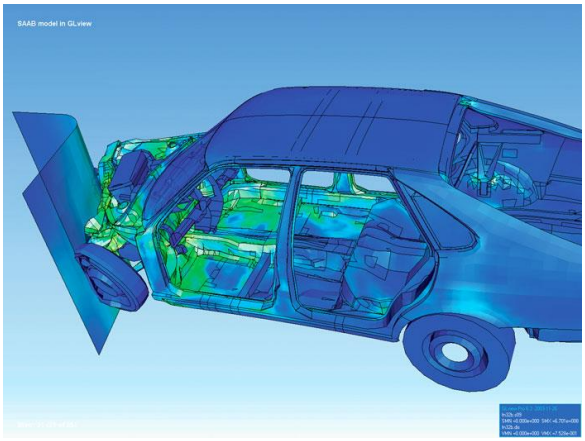
Implementasi penyimpanan tersatu beserta dengan penampil citra 3D akan membantu dalam pengayaan pengguna luas, seperti masyarakat luas yang ingin melihat dan mengetahui benda-benda bersejarah dengan mudah, atau masyarakat yang ingin berpartisipasi dalam restorasi benda bersejarah dengan mengambil model 3D untuk kemudia digubah menjadi reka ulang restorasi benda. Selain itu, implementasi ini akan menarik bagi kurator yang ingin menyatukan pendataan benda bersejarah bersamaan dengan preservasi benda tersebut untuk dapat memudahkan pembukuan dan beban kerja.

BAB 2
DASAR TEORI

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

2.1. Visualisasi

Visualisasi merupakan teknik untuk membuat gambar, diagram, atau animasi untuk menyampaikan suatu pesan. Melalui penggambaran visual, komunikasi baik untuk ide abstrak maupun konkrit dapat tercapai dari zaman dahulu kala. Visualisasi sekarang memiliki pengaplikasian yang sangat luas di ilmu pengetahuan, edukasi, teknik, multimedia interaktif, dan lain lain. Ada tiga tipe visualisasi, yaitu informasi, ilmiah, dan perangkat lunak.



Gambar 2.1 Penggunaan visualisasi dalam tabrakan mobil

Gambar 2.1 merupakan contoh penggunaan visualisasi dalam analisis tabrakan mobil. Bentuk deformasi citra 3D mobil yang telah memiliki titik-titik simulasi struktur dalam tabrakan dianalisis

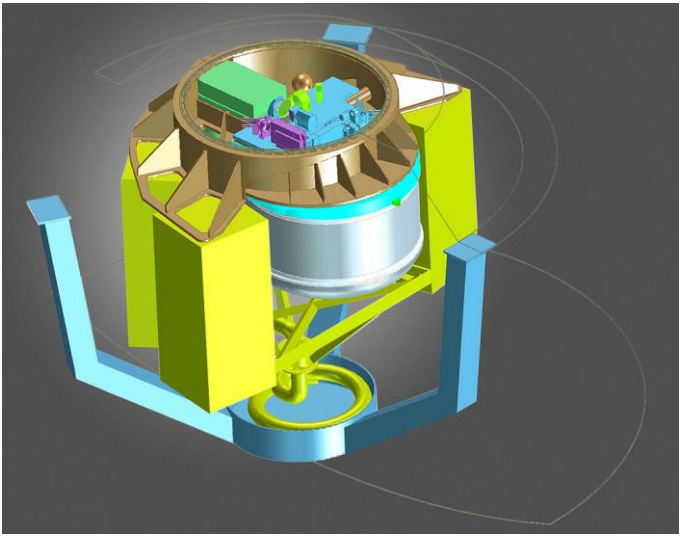
menggunakan *finite element analysis*, dengan titik yang terpengaruh dalam model memiliki visualisasi yang berbeda.

Visualisasi informasi didefinisikan sebagai penggunaan representasi visual interaktif terdukung komputer dari data abstrak untuk membantu pengetahuan. Ini dapat digunakan di hampir semua bidang untuk membantu pengguna pengurus problem yang ada di bidangnya. Teknik ini mulai pindah dari lab ke masyarakat dan pentingnya bidang ini terus bertambah. Dua bentuk visualisasi lainnya adalah ilmiah dan perangkat lunak, yang mengikuti prinsip yang serupa. Visualisasi ilmiah dideskripsikan sebagai ketika grafika komputer diaplikasikan ke data ilmiah untuk mendapatkan wawasan, menguji hipotesis, dan edukasi, sementara visualisasi perangkat lunak digunakan untuk pengertian sistem perangkat lunak kompleks dan siklusnya.

Visualisasi mengambil data dari kumpulan dan memprosesnya menjadi sesuatu yang dapat diproses dalam kognisi dan pengertian dari pengguna. Kontrol hanya dapat diambil pada saat proses visualisasi, dengan harapan pengguna yang memproses visualisasi dapat mengerti dan menginterpretasi dengan sesukses mungkin. Karena itu, visualisasi berusaha untuk memetakan data ke struktur tertentu dan menunjukkan data tersebut dengan sebaik mungkin

2.2. Objek 3D

Untuk visualisasi suatu objek, dibutuhkan cara untuk mendapatkan citra dari objek dalam dunia nyata dan memindahkannya ke dalam komputer untuk dapat divisualisasikan. Objek model 3D adalah suatu representasi matematis suatu objek dunia nyata dalam tiga dimensi. Model tersebut merepresentasikan benda fisik menggunakan kumpulan titik dalam lingkungan tiga dimensi, yang dihubungkan oleh entitas geometris seperti *triangle*, *line*, permukaan kurva, dan lain-lain. Model 3D dapat dibuat dengan tangan, secara algoritmis dalam proses *procedural modeling*, atau dengan pemindaian. Permukaan model dapat didefinisikan lebih lanjut menggunakan tekstur.



Gambar 2.2 Model 3D spektograf

Gambar 2.2 merupakan representasi model 3D dari spektograf dan kamera resolusi tinggi ERIS, yang diperuntukkan untuk Very Large Telescope yang dimiliki ESO. Model ini diperuntukkan sebagai desain konsep untuk konstruksi peralatan, sengan bagian-bagian model diwarnai berbeda untuk menunjukkan setiap komponen yang menyusun peralatan tersebut.

Model 3D dapat dibagi menjadi dua kategori, yaitu *solid* dan *shell*.

1. Model **solid**, atau padat, merupakan model yang mendefinisikan volume objek yang direpresentasikan. Model padat biasanya digunakan dalam simulasi teknik dan medis, dan biasanya dibuat dengan geometri padat konstruktif.
2. Model ***shell***, atau cangkang, merupakan model yang merepresentasikan permukaan suatu objek tanpa memperhatikan kepadatannya. Hampir semua model visual yang digunakan dalam *game* dan film adalah model cangkang.

Gambar 2.3 Render model Utah Teapot

Gambar 2.3 merupakan hasil *render* dari model *Utah Teapot*, pertama kali dibuat pada tahun 1975. Model *Utah Teapot* merupakan salah satu model yang paling sering digunakan dalam pendidikan 3D *modeling*. Model ini pertama kali dibuat oleh Martin Newell dan dimodifikasi oleh koleganya, Jim Blinn di 1975

Ada tiga cara popular untuk merepresentasikan suatu model, sebagai berikut.

1. *Polygonal modeling*, dimana titik-titik dalam lingkungan 3D, disebut verteks, dihubungkan dengan segmen-segmen garis untuk membentuk suatu *mesh* polygonal. Kebanyakan model 3D sekarang dibangun sebagai model polygonal bertekstur, karena lebih fleksibel dan perangkat dapat merender dengan lebih cepat. Namun, polygon bersifat planar dan hanya dapat mengira-ngira permukaan lengkung dengan menggunakan banyak polygon.
2. *Curve modeling*, dimana permukaan didefinisikan menggunakan kurva yang diatur dengan titik control berbeban. Kurva tersebut akan mengikuti titik-titik yang didefinisikan. Menaikkan beban suatu titik akan menarik kurva lebih dekat ke titik tersebut. Jenis kurva antara lain adalah *nonuniform rational B-spline* (NURBS), *spline*, *patch*, dan *geometric primitive*.

3. *Digital sculpting*, atau ukiran digital. Ada tiga tipe *digital sculpting* sebagai berikut.
    a. *Displacement*, dimana suatu model padat menyimpan posisi verteks baru menggunakan *image map* 32-bit yang menyimpan posisi yang berubah.
    b. *Volumetric*, yang berbasis sistem *voxel*, memiliki kapabilitas yang serupa dengan *displacement* namun tidak memiliki peregangan polygon ketika polygon yang ada dalam suatu wilayah tidak cukup untuk mendpatkan pembentukan yang diinginkan.
    c. *Dynamic tessellation* mirip dengan sistem voxel namun membagi permukaan mengunakan triangulasi untuk menjaga permukaan tetap mulus dan memperkenankan detil yang lebih baik.

Efek fotorealistik dalam 3D sering dicapai tanpa model *wireframe* dan kadang sulit dibedakan dengan benda yang asli. Keuntungan model 3D *wireframe* dari metode 2D antara lain adalah sebagai berikut.

1. Fleksibilitas, yaitu kemampuan untuk mengubah sudut penglihatan atau menganimasikan gambar dengan *rendering* perubahan yang lebih cepat.
2. Kemudahan *rendering*, seperti kalkulasi otomatis dan pemunculan efek fotorealistis secara langsung tanpa perkiraan.
3. Fotorealisme akurat, kurangnya kesalahan manusia seperti salah menaruh, terlalu banyak atau lupa untuk menambahkan efek tertentu.

Kekurangan dibandingkan dnegan metode 2D antara lain adalah butuhnya pendidikan penggunaan perangkat lunak dan kesulitan mencapai efek tertentu. Beberapa efek tertentu dapat dicapai menggunakan filter yang ada dalam perangkat lunak *3D modelling* yang digunakan.

2.3. 3D *Rendering*

3D *rendering* merupakan proses konversi model 3D menjadi gambar 2D dengan efek-efek tertentu dalam suatu komputer. Proses *rendering* merupakan proses terakhir dalam membuat citra atau animasi 2D yang diinginkan dari *scene* yang dibuat. Proses ini dapat dikomparasikan dengan mengambil foto atau merekam *scene* setelah persiapan dilakukan di dunia nyata. *Rendering* dapat memakan waktu dari sebagian detik hingga beberapa hari untuk satu citra atau *frame*. Metode yang berbeda akan lebih sesuai untuk *rendering* fotorealistis atau *real-time rendering*.

2.3.1. *Real-time rendering*

Proses *rendering* untuk media interaktif seperti *game* dan simulasi dikalkulasikan dan ditampilkan dalam *real-time* dengan rentang 20 hingga 120 *frame* per detik. Dalam *rendering* jenis ini, tujuannya adalah untuk menunjukkan sebanyak mungkin informasi ke mata pengguna dalam sebagian detik, yaitu dalam satu *frame*. Dalam kasus animasi 30 *frame* per detik, satu *frame* berarti memiliki waktu 1/30 detik. Tujuan utama adalah mencapai tingkat fotorealisme setinggi mungkin dalam kecepatan *rendering* yang dapat dimaklumkan, dengan tingkat yang biasanya ingin dicapai adalah 24 *frame* per detik sebagai tingkat minimum mata manusia dapat melihat ilusi pergerakan. *Rendering* jenis ini adalah jenis yang paling sering digunakan dalam *game*, dunia interaktif, dan lain-lain.

Gambar 2.4 Contoh *real-time rendering*

Gambar 2.4 merupakan *screenshot* dari *Second Life*, sebuah *online game* yang menggunakan *real-time rendering* untuk menampilkan objek-objek yang dilihat pemain. Setiap objek yang jatuh dalam *line-of-sight* dari pemain akan di-*render* oleh *game* unutk ditampilkan dalam bingkai permainan pemain.

2.3.2. *Non-realtime rendering*

Animasi untuk media noninteraktif seperti film dan video akan di-render jauh lebih lambat. Proses non-realtime akan membantu daya proses yang lebih lemah untuk mencapat tingkat kualitas citra yang lebih tinggi. Waktu rendering untuk satu frame individual dapat merentang dari beberapa detik hingga beberapa hari intuk scene yang sangat kompleks. Frame yang telah diproses akan disimpan dalam media penyimpanan yang akan dapat dipnidahkan ke media lain, seperti rol film dan media optik. Frame-frame ini kemudian akan ditampilkan secara sekuensial dengan framerate yang tinggi, dengan kecepatan standar antara 24, 25, atau 30 frame per detik untuk membuat ilusi pergerakan.

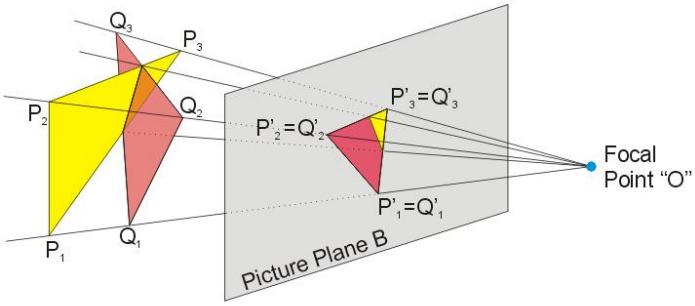
2.3.3. Model Refleksi dan *Shading*

Model refleksi dan *shading* digunakan unutk mendeskripsikan penampilan permukaan model. Grafik komputer 3D modern biasanya bergantung pada model refleksi sederhana yang disebut *Phong reflection model*. Dalam refraksi cahaya, sebuah konsep yang harus dipelajari adalah indeks refraksi. Dalam kebanyakan implementasi program 3D, nilai ini disebut sebagai indeks refraksi. *Shading* dapat dibagi menjadi dua teknik yang biasanya dipelajari secara terpisah.

1. *Surface shading*, yaitu bagaimana cahaya menyebar di atas suatu permukaan. Deskripsi model seperti ini biadanya dilakukan dengan suatu program yang disebut *shader*. Contoh penggunaan *shading* antara lain adalah pemetaan tekstur, yang menggunakan suatu sitra untuk menspesifikasikan warna *diffuse* untuk setiap titik di permukaan, yang akan memberikan detil yang lebih baik.
2. Refleksi, atau *scattering*, yaitu bagaimana cahaya berinteraksi dengan suatu permukaan di titik tertentu.

2.3.4. Proyeksi 3D

Objek 3D yang telah diproses harus dipipihkan agar perangkat tampilan seperti monitor dapat menampilkannya dalam hanya dua dimensi. Proyeksi 3D merupakan metode untuk memetakan objek tiga dimensi ke dalam lingkungan dua dimensi. Karena kebanyakan metode untuk menampilkan data grafis adalah berbasi pada media planar dua dimensi, proyeksi jenis ini tersebar luas, seperti di grafik komputer, bidang teknik dan *drafting.*

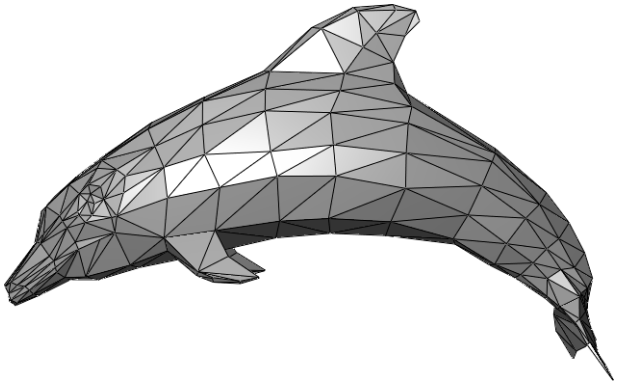


Gambar 2.5 Proyeksi perspektif

Gambar 2.5 merupakan contoh proyeksi perspektif, dimana titik fokus dan *viewpoint* kamera akan menentukan bagaimana objek akan terlihat dalam planar. Proyeksi perspektif akan membuat objek yang lebih jauh dari titik fokus terlihat lebih kecil, sesuai dengan perspektif titik.

2.4. Komponen Model 3D

Model 3D yang menggunakan polygon akan memiliki suatu *polygon mesh* sebagai representasi bentuk model. Suatu *polygon mesh* merupakan suatu kumpulan verteks, *edge* dan muka yang mendefinisikan bentuk objek polyhedral dalam model solid 3D dalam grafik komputer 3D. Muka objek biasanya akan terdiri dari segitiga, dalam bentuk *triangle mesh*, segiempat, atau bentuk polygon konveks lainnya. Hal ini akan mempermudah dalam proses *rendering*.



Gambar 2.6 Contoh *mesh* untuk model lumba-lumba

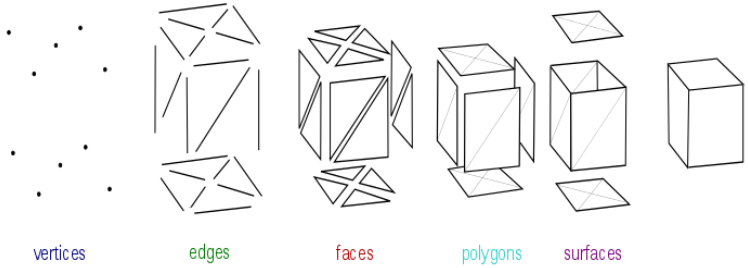
Gambar 2.7 merupakan contoh bagaimana *mesh* untuk suatu model akan terlihat, dengan contoh model lumba-lumba. *Mesh* yang digunakan merupakan *triangle mesh*, dengan segitiga menyusun permukaan model dan membentuk model secara keseluruhan.

Objek yang dibentuk dengan menggunakan *polygon mesh* harus dapat menyimpan beberapa tipe elemen, seperti verteks, *edge*, *faces*, polygon, dan permukaan. Dalam beberapa aplikasi, hanya verteks, *edge*, dan antara polygon atau permukaan yang akan disimpan. Suatu renderer mungkin hanya mendukung *face* yang bersisi tiga, maka polygon harus dibuat menggunakan *face* tersebut. Namun, banyak *renderer* yang mendukung *quads* atau polygon bersisi lebih banyak, atau dapat mengkonversi polygon ke *triangle* dalam proses. Berikut ini adalah definisi setiap elemen.

1. Verteks, merupakan posisi titik dalam lingkungan tiga dimensi yang memiliki informasi lain seperti warna, vektor normal, dan koordinat tekstur.
2. *Edge*, merupakan hubungan antara dua verteks.
3. *Face*, merupakan kumpulan set *edge* yang tertutup. Sebuah *triangle face* memiliki tiga *edge*, dan sebuah *quad face* memiliki empat *edge*. Sebuah polygon merupakan suatu set *face* yang koplanar dengan satu sama lain. Dalam sistem yang mendukung

multi-sided faces, polygon dan *face* memiliki arti yang sama. Namun, kebanyakan perangkat *rendering* hanya memiliki dukungan untuk *face* dengan tiga atau empat sisi, sehingga pilogon akan direpresentasikan sebagai beberapa *face*.

4. *Surface*, atau permukaan, lebih sering disebut sebagai *smoothing group*. Elemen ini digunakan untuk mengelompokkan *face* yang dapat dihaluskan dengan algoritma *shading*.
5. Grup, mendefinisikan bagian *mesh* yang terpisah, digunakan untuk mendefinisikan objek untuk *skeletal animation* atau aktor yang berbeda untuk *non-skeletal animation*.
6. Material, yang akan didefinisikan untuk membantu bagian *mesh* yang berbeda untuk menggunakan *shader* yang berbeda.
7. Koordinat UV, yang didukung beberapa format *mesh*, merupakan representasi 2D *mesh* yang telah dibuka lebar, untuk menunjukkan bagian peta tekstur mana yang diaplikasikan ke polygon tertentu. Sebuah *mesh* juga dapat memiliki atribut verteks lain seperti warna, vektor tangen, *weight maps*, dan lain-lain.



Gambar 2.7 Elemen suatu model

Gambar 2.7 merupakan representasi setiap elemen dalam suatu model sederhana. Verteks saling menghubungkan menjadi *edge*, yang saling menghubungkan menjadi *face*, yang dikelompokkan menjadi polygon dan permukaan.

Representasi suatu *polygon mesh* dapat dilakukan dalam beberapa cara, yang menggunakan cara berbeda untuk menyimpan data verteks, *edge*, dan *face*.

1. *Face-vertex*, hanya menggunakan daftar verteks, dan set polygon yang menunjuk ke verteks yang digunakan.
2. *Winged-edge*, dimana setiap *edge* merujuk ke dua verteks, dua *face*, dan empat *edge* yang menyentuhnya.
3. *Half-edge*, mirip dengan sistem *winged-edge* namun hanya setengah informasi *traversal* yang digunakan.
4. *Quad-edge*, menyimpan *edge*, *half-edge*, dan verteks tanpa referensi ke polygon. Poligon akan bersifat implisit dalam representasi dan dapat didapatkan dengan berjalan melalui struktur.
5. *Corner-table*, menyimpan verteks dalam tabel yang disiapkan, sehingga berjalam melalui tabel secara implisit mendefinisikan polygon.
6. *Vertex-vertex*, hanya merepresentasikan verteks yang merujuk ke verteks lain, dengan *edge* dan *face* bersifat implisit.

2.5. Wavefront OBJ

OBJ atau .OBJ adalah format *file* definisi geometri yang pertama kali dikembangkan oleh Wavefront Technologies. Format ini terbuka dan telah diadopsi oleh vendor aplikasi 3D lain, dan merupakan format yang diterima secara universal. Format ini merupakan format *simple-data* yang merepresentasikan geometri 3D saja, yaitu posisi setiap vertex, posisi UV dari setiap vertex koordinat tekstur, *normal* dari vertex, dan mukaan yang membuat setiap polygon didefinisikan sebagai daftar vertex dan vertex tekstur. Verteks disimpan berlawanan arah jarum jam dan tidak perlu menyimpan *normal* dari mukaan secara eksplisit. Koordinat OBJ tidak memiliki unit, namun *file* OBJ dapat memiliki informasi untuk skala dalam baris komentar yang dapat dibaca manusia.

Dalam suatu *file* OBJ, terdapat beberapa *identifier* yang melambangkan data-data yang direpresentasikan dalam *file* OBJ, antara lain sebagai berikut.

1. Verteks geometris
   Dilambangkan dengan v , merupakan titik verteks dalam ruang

3D, menggunakan koordinat `(x,y,z,w)` dengan `w` opsional dan *default* ke nilai 1.0. Beberapa aplikasi juga mendukung warna verteks, dengan menggunaan nilai RGB setelah entri vertex.

2. Verteks parameter ruangan
   Dilambangkan dengan `vp` , merupakan koordinat parameter verteks `(u,v,w)` dengan `w` opsional.
3. Verteks tekstur
   Dilambangkan dengan `vt` , merupakan koordinat tekstur `(u,v,w)` dengan `w` opsional
4. Normal verteks
   `vn` , melambangkan titik normal `(i,j,k)` . mempengaruhi *shading* dan *rendering* permukaan geometri.
5. *Face*/permukaan
   Dilambangkan dengan `f` , terdiri dari vertex/tekstur/normal, dengan terstur dan normal opsional
6. `mtllib`
   Melambangkan arah untuk material objek yang tersimpan di *file* MTL.
7. `o`
   Melambangkan nama objek.
8. `g`
   Melambangkan grup objek.
9. `usemtl`
   Melambangkan material yang akan digunakan hingga baris ini muncul lagi

```
# List of geometric vertices, with (x,y,z[,w]) coordinates, w is
optional and defaults to 1.0.
v 0.123 0.234 0.345 1.0
v ...
...
# List of texture coordinates, in (u, v [,w]) coordinates, these
will vary between 0 and 1, w is optional and defaults to 0.
vt 0.500 1 [0]
vt ...
...
# List of vertex normals in (x,y,z) form; normals might not be
unit vectors.
vn 0.707 0.000 0.707
```

```
vn ...
...
# Parameter space vertices in ( u [,v] [,w] ) form; free form
geometry statement ( see below )
vp 0.310000 3.210000 2.100000
vp ...
...
# Polygonal face element (see below)
f 1 2 3
f 3/1 4/2 5/3
f 6/4/1 3/5/3 7/6/5
f 7//1 8//2 9//3
f ...
...
```

Kode 2.1 Format berkas OBJ

Kode 2.1 merupakan contoh format untuk suatu berkas OBJ sederhana. Baris yang diawali dengan symbol # adallah baris komentar, dan tidak dihitung dalam proses pembacaan berkas. Setiap baris memiliki nama baris di depan, dan isi baris, dibatasi dengan spasi. Format berkas adalah *plaintext*, yang berarti berkas dapat dibuka di *text editor* biasa.

Format .MTL adalah format *file* rekanan yang mendeskripsikan sifat *surface shading* atau material dari objek dalam satu atau lebih *file* OBJ. *File* OBJ mereferensi ke satu atau lebih *file* MTL yang disebut *material library* dan merefrensikan satu atau lebih deksripsi material dari namanya. Format ini mendefinisikan sifat pantulan cahaya untuk *rendering* berdasarkan model pantulan Phong. Standar ini sudah didukung oleh banyak paket perangkat lunak dan berguna untuk pertukaran material.

Seperti dalam *file* OBJ, *file* MTLmemiliki beberapa *identifier* yang melambangkan data-data yang direpresentasikan dalam *file* MTL, antara lain sebagai berikut.

1. `newmtl`
   Melambangkan mulainya material baru
2. `Ka`
   Melambangkan *ambient color* `(r,g,b)`

3. `Kd`
   Melambangkan *diffuse color* `(r,g,b)`
4. `Ks`
   Melambangkan *specular color* `(r,g,b)`
5. `illum`
   Melambangkan model penerangan, dimana 1 merupakan material datar tanpa *specular highlight* dan 2 menyatakan adanya *specular highlights*
6. `Ns`
   Melambangkan kilau material
7. `d or Tr`
   Melambangkan transparensi material
8. `map_Ka/Kd`
   Melambangkan nama tekstur

```
newmtl Textured
Ka 1.000 1.000 1.000
Kd 1.000 1.000 1.000
Ks 0.000 0.000 0.000
d 1.0
illum 2
map_Ka lemur.tga           # the ambient texture map
map_Kd lemur.tga           # the diffuse texture map (most of the
                           # time, it will be the same as the
                           # ambient texture map)
map_Ks lemur.tga           # specular color texture map
map_Ns lemur_spec.tga      # specular highlight component
map_d lemur_alpha.tga      # the alpha texture map
map_bump lemur_bump.tga    # some implementations use 'map_bump'
                           # instead of 'bump' below
bump lemur_bump.tga        # bump map (which by default uses
                           # luminance channel of the image)
disp lemur_disp.tga        # displacement map
decal lemur_stencil.tga    # stencil decal texture (defaults to
                           # 'matte' channel of the image)
```

Kode 2.2 Format Berkas MTL

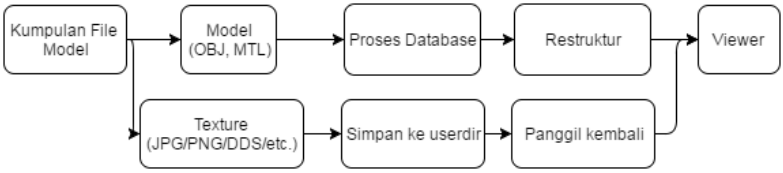
Kode 2.2 merupakan representasi dasar berkas MTL, dengan beberapa fitur yang akan digunakan oleh *renderer* dalam menampilkan model. Setiap elemen entri menunjukka suatu fitur tertentu yang ditunjukkan dalam material objek 3D. Entri dalam berkas MTL akan dipetakan ke dalam objek dalam berkas OBJ untuk memberikan atribut tertentu untuk setiap objek.

Halaman ini sengaja dikosongkan

BAB 3
DESAIN SISTEM DAN IMPLEMENTASI

3.1. Desain Sistem

Tugas akhir ini bertujuan untuk membuat sistem yang mampu untuk menerima kumpulan berkas yang merepresentasikan suatu model 3D dan mengumpulkan sedemikian rupa sehingga dapat mengambil kembali kumpulan tersebut ataupun menampilkan kembali model tersebut dalam suatu *viewer*. Kumpulan ini memiliki jenis berkas antara lain OBJ, MTL, dan format penyimpanan gambar tekstur seperti JPG, PNG, DDS, dan lain-lain. Untuk berkas OBJ dan MTL, kedua berkas tersebut adalah *plaintext* yang dapat dibuka dengan mudah, dan diputuskan untuk emnyimpan kedua format tersebut dalam *database*. Berkas tekstur akan disimpan di folder lain yang dikategorikan sesuai pengunggah yang akan dipanggil bersamaan dengan *database* berisi informasi model untuk direkonstruksi menjadi informasi yang bisa diproses oleh *viewer*.



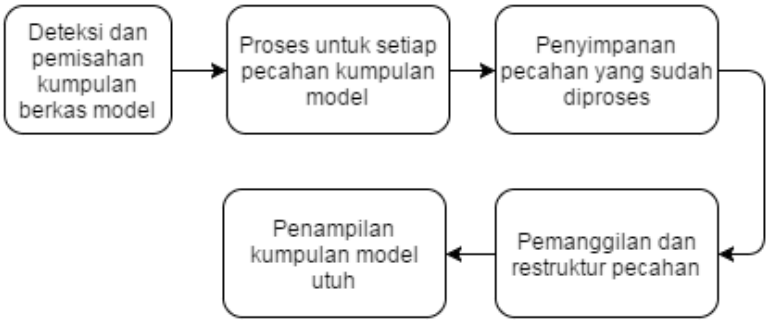
Gambar 3.1 Desain Sistem secara umum

Gambar 3.1 merupakan alur desain kerja sistem secara umum. Pemisahan berkas akan terjadi sebelum proses ke dalam *database*, dan setiap berkas akan diproses secara sekuensial. Penyimpanan ke dalam *database* akan hanya berlaku untuk berkas OBJ dan MTL, dengan berkas tekstur akan disimpan dalam folder milik pengguna, dibatasi sesuai dengan nama model yang diunggah oleh pengguna. Pemanggilan kembali akan membuat berkas yang tersimpan dalam *database* untuk direkonstruksi kembali sementara hingga penampilan selesai.

3.2. Alur Kerja

Kumpulan berkas model memiliki ident tersendiri dalam setiap berkas yang akan digunakan dalam pemisahan. Proses unggah bertujuan untuk memuat kumpulan ke dalam sistem yang selanjutnya akan dipisah dan diproses sesuai tipe berkas. Berkas model, yaitu format OBJ dan MTL akan dipisahkan dari kumpulan untuk diproses dan dimassukan ke dalam *database*, sementara berkas *texture*, yaitu format JPG, PNG, DDS, dan lain-lain akan diproses untuk disimpan pada direktori sesuai pengunggah, dan didata pada *database* untuk persiapan pemanggilan.

Berkas model akan dipecah dan dimasukkan kedalam *database* sesuai dengan tipenya, dengan berkas OBJ dipecah sesuai dengan grup dalam model dan berkas MTL dipecah sesuai dengan material. Ketika diperlukan, berkas model akan direstruktur seperti semula dan diberikan ke *viewer* beserta dengan berkas tekstur untuk dibuat menjadi grafik 3D siap tampil. Gambaran alur kerja sistem akan dijelaskan dalam gambar berikut.

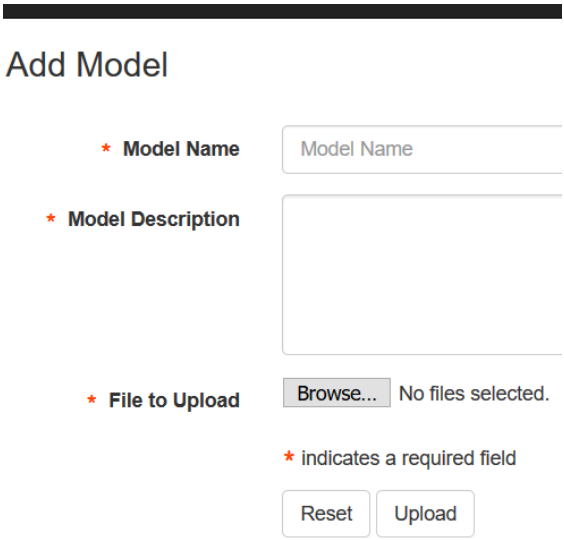


Gambar 3.2 Alur Kerja Sistem

Gambar 3.2 merupakan alur kerja dalam sistem ini. Pemanggilan pecahan akan dilakukan ketika sistem menerima permintaan penampilan suatu model dari pengguna. Penampilan akan dilakukan ketika semua pecahan telah diterima dari *database* dan direkonstruksi ulang menjadi format yang dapat dibaca oleh *renderer*.

3.3. Pengunggahan Model

Pengunggahan berkas model dilakukan dalam bentuk satu *folder* yang akan diunggah ke *database*. Berikut adalah form unggah model yang digunakan.



Gambar 3.3 Form Pengunggahan Model

Gambar 3.3 merupakan bentuk dasar *form* pengunggahan model. Tanda asterisk merah menandakan wilayah yang harus diisi untuk dapat mengunggah model, untuk mencegah *spamming* model tanpa deskripsi. Hanya pengguna yang telah *login* yang dapat mengakses lembar ini.

Pengunggahan menerima nama model, deskripsi model, dan berkas-berkas yang digunakan sebagai input, dan akan memberikan input ke proses deteksi dan pemisahan. Input hanya akan menerima berkas JPEG, PNG, BMP, dan DDS sebagai input tekstur model. Ada kendala dalam implementasi pengunggahan berkas yang berasal dari implementasi *browser* yang membuat pengunggahan tidak bisa melebihi 20 berkas secara sekaligus.

3.4. Deteksi dan Pemisahan Kumpulan Berkas Model

Deteksi elemen kumpulan berkas dilakukan pada saat pengunggahan, dengan memisahkan antara berkas model dengan berkas tekstur. Pemisahan dibuat sesuai dengan format setiap berkas yang berhasil diunggah oleh pengguna. Alur sistem untuk tahap ini adalah sebagai berikut.

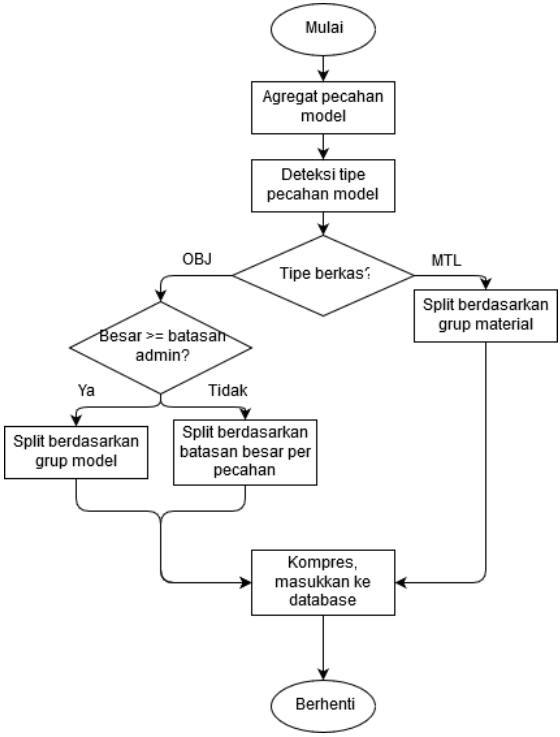
Gambar 3.4 Alur sistem pemisahan data

Gambar 3.4 merupakan alur sistem untuk proses pemisahan data. Pemisahan dalam tahap ini akan mengatur bagaimana antrian berkas akan berjalan. Kedua alur berkas akan melaju ke proses yang sama, namun dengan tempat selesai yang berbeda.

Pengunggahan dilakukan oleh pengguna teregistrasi, dengan entri unggahan akan dibuat untuk satu kali unggah, berisi tentang nama model, deskripsi model, dan data nama OBJ dan MTL model. Setiap pecahan data akan ditandai dengan ident pengguna dan entri model, yang akan membanti mengidentifikasi pecahan tersebut. Pecahan kemudian akan dimasukkan ke dalam proses sesuai dengan tipe pecahan tersebut, dimana pecahan model akan mengarah ke proses *database* dan pecahan tekstur akan mengarah ke proses unggah ke direktori pengguna.

3.5. Proses untuk Setiap Pecahan Model

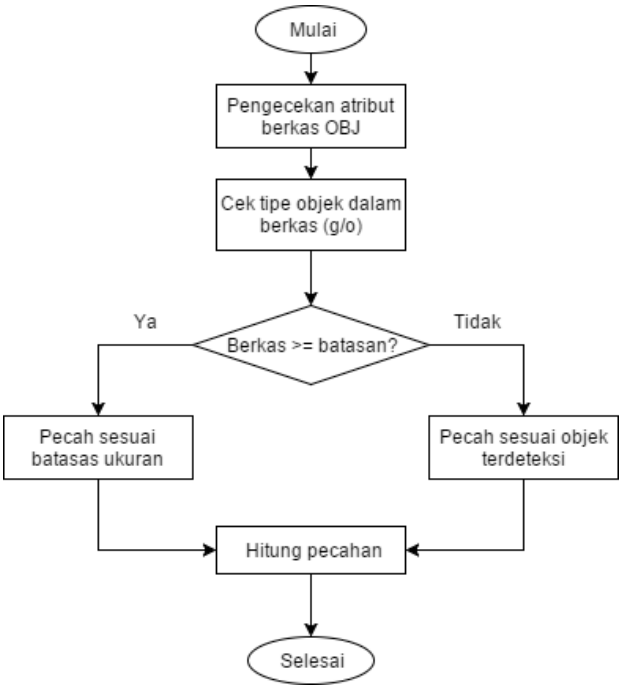
Proses untuk setiap pecahan kelompok akan berbeda, tergantung dari tipenya. Alur sistem dari kedua proses itu adalah sebagai berikut.



Gambar 3.5 Alur sistem proses untuk pecahan model

Gambar 3.5 merupakan grafik alur sistem untuk proses dalam setiap pecahan model. Deteksi pemisahan model akan menggunakan ekstensi berkas sebagai penanda bagaimana berkas akan diproses. Deteksi pemisahan juga akan mendeteksi bagaimana berkas akan dipecah, dengan mengukur seberapa besar setiap berkas yang masuk ke dalam proses.

Pecahan model akan diproses ke *database*, sementara pecahan tekstur akan diproses ke direktori pengguna. Pecahan bertipe OBJ akan dipecah sesuai dengan ukuran dari berkas, dengan berkas yang berukuran kurang dari besaran yang telah diterapkan admin akan dipecah menurut penentuan grup dalam model, yang ditandai dengan notasi g. Jika berkas lebih besar dari ukuran batasan, berkas akan dipecah sesuai dengan ukuran dari besaran yang diatur oleh admin. Alur dan pseudokode untuk fase pemecahan ini adalah sebagai berikut.



Gambar 3.6 Alur sistem pemecahan berkas OBJ

Gambar 3.6 merupakan alur sistem dalam pemecahan berkas OBJ. Pemecahan akan mengambil data besar berkas dari proses seperti dalam Gambar 3.5, dan akan memecah sesuai dengan besar berkas. Setelah dipecah, berkas akan dihitung untuk memastikan semua isi berkas ada, dan untuk membantu dalam proses seperti dalam Gambar 3.7.

```
// Process for .obj files
if (file_extension === "obj") {
  $filesize_obj   = filesize ($temp_path);    // get filesize
  $file_obj       = file ($temp_path);        // get file
  $file_json      = encode($file_obj);   // encode to json

  // check if file is using g/o for object declaration, set up
  fixer
  $fixer = setFixer($file_json, $filesize_obj);

  // explode file
  if ($filesize_obj >= $cutsize) {
    $obj_chop = explode_by_cutsize($file_json); // by cutsize
  }
  else {
    if ($fixer_obj !== "") {
      $obj_chop = explode_by_group($file_json);
    }
    else {
      $obj_chop = explode_by_cutsize($file_json);
      // by cutsize
    }
  }

  // count for exploded pieces
  $headcount = count($obj_chop);

  // repair newline
  $piece_obj = setNewline($obj_chop);
```

Kode 3.1 Pemecahan dan konversi OBJ

Kode 3.1 merupakan pseudokode untuk pemecahan dan konversi OBJ. Pemecahan memiliki tiga tahap dasar, yaitu tahap deteksi, tahap pemecahan, dan tahap sensus. Taham pemecahan menggunakan data besar berkas dari tahap deteksi dan batasan besar yang ditatapkan sebelumnya untuk memecah berkas.

Deteksi tipe berkas juga akan mencatat besar berkas, sebagai perbandingan dengan batas besar yang ditetapkan sebelumnya. Data berkas akan dikonversi preliminer menjadi satu string JSON sebelum diproses. Deteksi objek dalam berkas dilakukan sebagai bahan untuk memperbaiki potongan string kemudian, yang disimpan dalam variable `$fixer_obj`. Pemotongan menggunakan fungsi `explode` yang menyelubungi fungsi `wordwrap` agar dapat mendapatkan potongan string yang tidak memotong di tengah entri objek. Hasil pemotongan kemudian akan dicatat jumlahnya dalam variabel `count_objpiece`.

Pecahan yang sudah dibagi akan diproses agar menjadi bentuk yang dapat disimpan, yaitu sebagai format JSON string, dan akan disimpan dalam database. Pemisahan akan dilakukan berdasarkan seberapa banyak pecahan yang diterima dari proses Kode 3.1 sebelumnya.

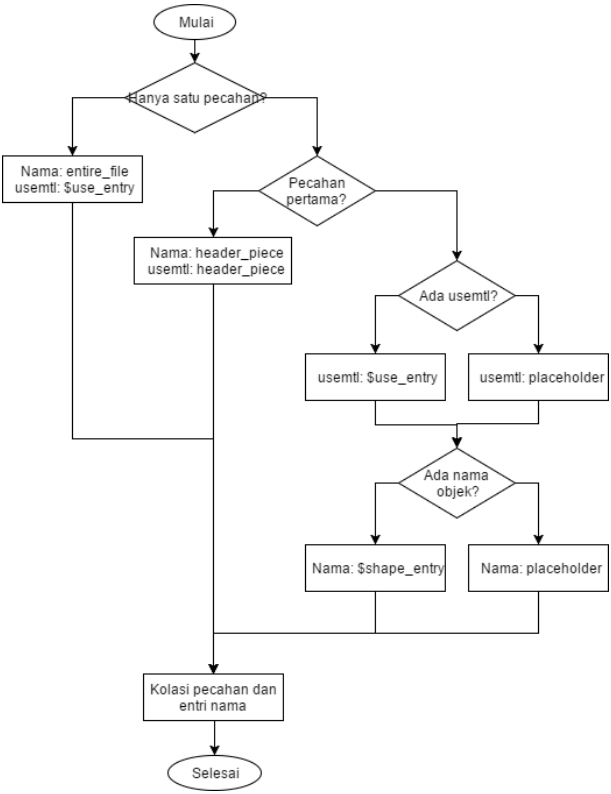
```
// process pieces for pushing to database, fix json
  foreach ($piece_obj as $key_obj => $objpiece_json) {
    // only 1 detected section
    if ($headcount <2) {
      if ($is_there_usemtl !== false) {
        $use_entry = get_usemtl();
      }

      $shape_entry[] = "entire_file";
      $obj_json[] = $objpiece_json;
    }
    // unused first entry in array
    else if ($first_in_line !== false) {
      $shape_entry[] = "header_piece";
      $use_entry[] = "header_piece" ;
      $obj_json[] = $objpiece_json;
    }
    // no other complications
    else {
      if (($is_there_usemtl !== false) {
        $use_entry = get_usemtl();
      }
      else {
        $use_entry = "--placeholder--";
      }

      if ($fixer_obj !== "") {
        if ($filesize_obj >= $cutsize) {
          $shape_entry = "--placeholder--";
```

```
        }
        else {
          $shape_entry = get_shape_name();
        }
      }
      else {
        $shape_entry = "--placeholder--";
      }
    }
  }

  // collate pieces to 1 var, unset unused vars
  $obj_collator = collateJson($obj_json);

  // output is $use_entry, $shape_entry and $obj_collator
```

Kode 3.2 Proses penamaan entri pecahan

Kode 3.2 merupakan pseudokode untuk penamaan pecahan berkas OBJ. Penamaan mendeteksi pecahan berdasarkan seberapa banyak pecahan yang dideteksi, sesuai dengan kode 3.1. Tahap ini juga menyediakan nama *placeholder* untuk kasus pengguna yang berada diluar yang dapat diambil, seperti pecahan yang tidak memiliki nama objek..

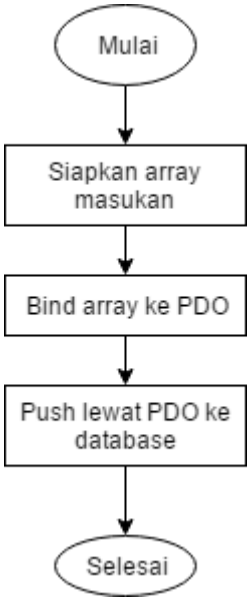
Pecahan akan dikelompokkan berdasarkan peran pecahan tersebut dalam berkas keseluruhan. Jika hanya terdeteksi satu pecahan dan tidak melebihi batasan pecahan yang ditetapkan, pecahan tersebut akan dilabeli sebagai berkas keseluruhan. Pecahan pertama akan selalu dilabeli sebagai pecahan *header*, untuk membantu jika perlu dilakukan perbaikan manual. Jika `usemtl`, yang menandakan material yang digunakan, dan nama bentuk dalam pecahan diketahui, maka entri material dan nama bentuk akan terisi. Jika sebaliknya, kedua entri tersebut hanya akan diisi `placeholder`. Keluaran dari proses ini adalah tiga *array* yang berisi nama untuk entri, nama objek yang diwakili entri, dan hasil potongan yang akan menjadi entri *database*.



Gambar 3.7 Alur penamaan pecahan OBJ

Gambar 3.7 merupakan alur penamaan pecahan OBJ yang menghasilkan pseudokode seperti pada Kode 3.2. Setiap pecahan akan melalui proses penamaan, dengan hasil paling terakhir akan dimasukkan ke dalam kolasi pecahan untuk disiapkan masuk kedalam *database.*

Hasil *array* yang didapatkan dari proses Kode 3.2 diatas kemudian akan diunggah ke dalam server *database*. Proses unggah akan dienkapsulasi dalam blok `try-catch` untuk membantu jika terjadi kesalahan dalam pemotongan. Kode untuk pembagian grup dan konversi ke JSON pada pecahan OBJ adalah sebagai berikut.



Gambar 3.8 Alur pengunggahan kolasi pecahan

Gambar 3.8 merupakan larus sistem dalam tahap pengunggahan hasil kolasi pecahan. Setiap pecahan akan diikat dalam transaksi PDO untuk kemudian didorong ke dalam *database*, dengan menjaga transaksi tetap memiliki provisi untuk gagal dengan benar.

```
try {
  $stmt_obj->execute(array(
    ':model_obj_name' => $name,
    ':id_model_list' => $modelId));
  foreach ($obj_collator as $key_obcl => $obj_input) {
    $stmt_obbl->bindValue(':obbl_id_model',
      $modelId, PDO::PARAM_INT);
    $stmt_obbl->bindValue(':obbl_group_name',
      $shape_entry[$key_obcl], PDO::PARAM_STR);
    $stmt_obbl->bindValue(':obbl_group_use',
      $use_entry[$key_obcl], PDO::PARAM_STR);
    $stmt_obbl->bindValue(':obbl_obj_blob',
      $obj_input, PDO::PARAM_STR);
    $stmt_obbl->execute();
```

```
    }
  }
  catch (PDOException $ex) {
    echo $ex->getMessage();
  }
}
```

Kode 3.3 Proses pengunggahan pecahan OBJ

Kode 3.3 merupakan implementasi alur pecahan untuk diunggah. Implementasi ini akan mengisi tabel `blob_obj` dengan pecahan yang sudah dibagi dan dikonvers serta ident untuk membantu dalam proses pengembalian seperti semula.

id_blob_obj	obbl_id_model	obbl_group_name	obbl_group_use	obbl_obj_blob
392	29	header_piece	header_piece	["# \r\n","# Wavefron
393	29	group1	Hgn_BattleCruiser_Main_EXPORT	["g group1\r\n","v -20
394	29	group2	Hgn_BattleCruiser_Engine_EXPORT	["g group2\r\n","v -17
395	29	group3	Hgn_BattleCruiser_Thruster_EXPO	["g group3\r\n","v 17.
396	29	group4	Hgn_BattleCruiser_Badge_EXPORT	["g group4\r\n","v 134
397	29	group11	Hgn_BattleCruiser_Main_EXPORT0	["g group11\r\n","v 66
398	29	group21	Hgn_BattleCruiser_Thruster_EXPO0	["g group21\r\n","v -4
399	29	group12	Hgn_BattleCruiser_Main_EXPORT1	["g group12\r\n","v 11
400	29	group13	Hgn_BattleCruiser_Main_EXPORT2	["g group13\r\n","v 11
401	29	group14	Hgn_BattleCruiser_Main_EXPORT3	["g group14\r\n","v -1
402	29	group15	Hgn_BattleCruiser_Main_EXPORT4	["g group15\r\n","v -1
403	29	group16	Hgn_BattleCruiserKineticBurstCa	["g group16\r\n","v -1
404	29	group17	Hgn_BattleCruiserKineticBurstCa2	["g group17\r\n","v 3.
405	29	group18	Hgn_BattleCruiserKineticBurstCa1	["g group18\r\n","v 9.

Gambar 3.9 Database penyimpanan, tabel blob_obj

Gambar 3.9 merupakan bentuk tabel penyimpanan pecahan OBJ. Pecahan dimasukkan dengan empat data kolom, yaitu ID pecahan, ID model pecahan, nama grup pecahan, nama material pecahan, dan pecahan itu sendiri, terkode dalam format JSON. Sistem dua ID ini akan membantu dalam rekonstruksi kembali model, karena hanya perlu memanggil kembali pecahan yang memiliki kode model tersebut.

Pecahan bertipe MTL akan dipecah menurut penentuan grup material, yang ditandai dengan notasi `newmtl`. Pecahan yang sudah dibagi akan diproses agar menjadi bentuk yang dapat disimpan, dalam proses ini adalah format JSON string, dan akan dikompres untuk disimpan dalam database. Kode untuk fase pemecahan ini adalah sebagai berikut.

Gambar 3.10 Alur pecahan berkas MTL

Gambar 3.10 merupakan rancangan alur sistem untuk pemecahan berkas MTL. Pemecahan akan mendeteksi berkas MTL dari berkas lain, dan akan memecah sesuai dengan material yang terdeteksi dalam berkas MTL.

```
// Process for .mtl files
elseif (pathinfo($name, PATHINFO_EXTENSION) === "mtl") {
  $fixer_mtl = "\"newmtl ";
  $file_mtl  = file ($temp_path);  // get file path
  $mtl_chop  = explode_mtl();
  $count_mtlpiece = count($mtl_chop);

  // repair newline
  $piece_mtl = setNewline($mtl_chop);
```

Kode 3.4 Deteksi dan pemecahan MTL

Kode 3.4 merupakan pseudokode pendeteksi dan pemecahan berkas MTL. Deteksi tipe berkas juga akan mencatat besar berkas, sebagai perbandingan dengan batas besar yang ditetapkan sebelumnya. Data berkas akan dikonversi preliminer menjadi satu string JSON sebelum

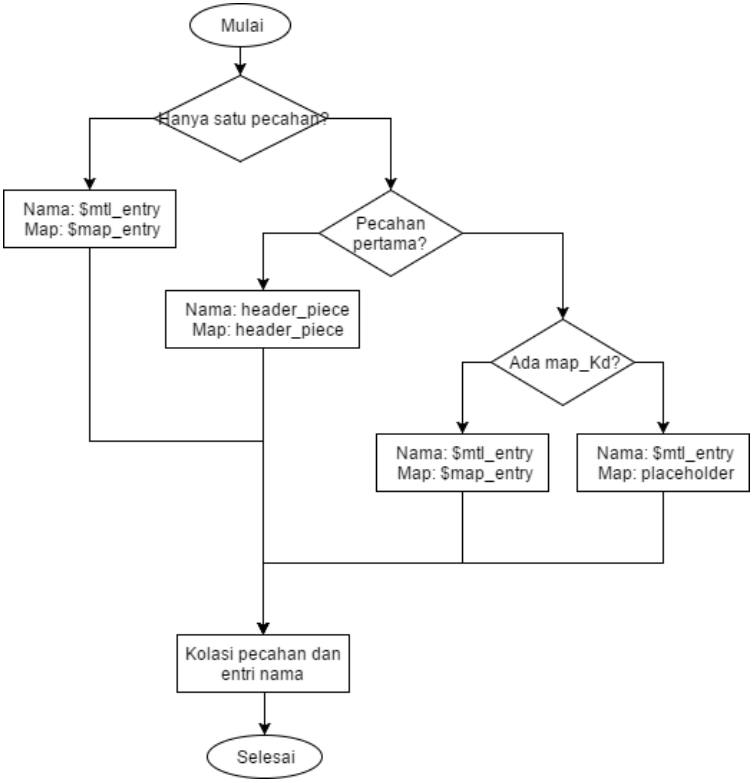
diproses. Deteksi material adalah sebagai bahan untuk memperbaiki potongan string kemudian Pemotongan menggunakan fungsi `explode` yang dapat mendapatkan potongan string yang tidak memotong di tengah entri objek. Hasil pemotongan kemudian akan dicatat jumlahnya dalam variabel `count_mtlpiece`.

Pecahan akan diproses agar menjadi bentuk JSON string, dan akan disimpan dalam database. Pemisahan akan dilakukan berdasarkan seberapa banyak pecahan yang diterima dari proses Kode 3.4 sebelumnya. Kode untuk pembagian grup dan konversi ke JSON pada pecahan MTL adalah sebagai berikut.

```
// fix json array entry
  foreach ($piece_mtl as $key_mtl => $mtlpiece_json) {
    if ($count_mtlpiece <2) {
      $mtl_entry = explode_by_name();

      $map_entry = explode_by_map();

      $mtl_json[] = $fixer_mtl.$mtlpiece_json;
    }
    else if ($key_mtl < 1) {
      $mtl_entry[] = "header_piece";
      $map_entry[] = "header_piece";
      $mtl_json[] = $mtlpiece_json;
    }
    else {
      $mtl_entry = explode_by_name();

      if ($is_there_map !== false) {
        $map_entry = explode_by_map();
      }
      else {
        $map_entry[$key_mtl] = "--placeholder--";
      }

      $mtl_json[] = $fixer_mtl.$mtlpiece_json;
    }
  }
  $mtl_collator = collateJson($mtl_json);
  // output is $mtl_entry, $map_entry and $json_collator
```

Kode 3.5 Pemecahan dan konversi MTL

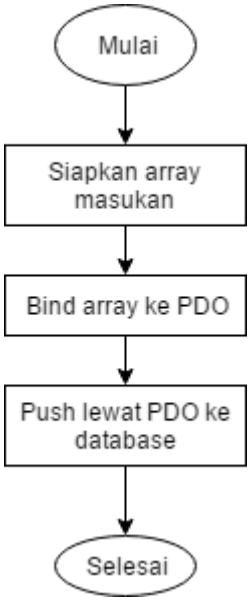
Kode 3.5 merupakan pseudokode pemecahan MTL. Pecahan akan dikelompokkan berdasarkan peran pecahan tersebut dalam berkas keseluruhan. Jika hanya terdeteksi satu pecahan, pecahan tersebut akan diproses sendiri dan diberi label entri sesuai dengan konten yang ada dalam pecahan. Jika terdapat lebih dari satu pecahan, pecahan pertama akan selalu dilabeli sebagai pecahan *header*, untuk membantu jika perlu dilakukan perbaikan manual. Deteksi akan memberikan label entri untuk nama material dan pemetaan material. Keluaran dari proses ini adalah tiga *array* yang berisi nama material, nama pemetaan material, dan hasil potongan yang akan menjadi entri *database*.



Gambar 3.11 Alur kerja pemisahan berkas MTL

Gambar 3.11 merupakan bagaimana alur kerja dalam pemisahan dan penamaan pecahan berkas MTL. Pecahan akan dinamai berdasarkan nama material yang diwakili dan nama objek yang mengambil material tersebut. Setelah dilakukan pemisahan, setiap pecahan akan dikolasi menjadi satu *array* untuk kemudian diunggah ke dalam *database*.

Hasil *array* yang didapatkan dari proses Kode 3.5 diatas kemudian akan diunggah ke dalam server *database*. Proses unggah akan dienkapsulasi dalam blok `try-catch` untuk membantu jika terjadi kesalahan dalam pemotongan. Kode untuk pembagian grup dan konversi ke JSON pada pecahan MTL adalah sebagai berikut.



Gambar 3.12 Alur pengunggahan kolasi pecahan

Gambar 3.12 merupakan arus sistem dalam tahap pengunggahan hasil kolasi pecahan. Setiap pecahan akan diikat dalam transaksi PDO untuk kemudian didorong ke dalam *database*, dengan menjaga transaksi tetap memiliki provisi untuk gagal dengan benar.

```
  try {
    $stmt_mtl->execute(array(
      ':model_mtl_name' => $name,
      ':id_model_list' => $modelId));
    foreach ($mtl_collator as $key_mtcl => $mtl_input) {
      $stmt_mtbl->bindValue(':mtbl_id_model',
        $modelId, PDO::PARAM_INT);
      $stmt_mtbl->bindValue(':mtbl_mat_name',
        $mtl_entry[$key_mtcl], PDO::PARAM_STR);
      $stmt_mtbl->bindValue(':mtbl_mat_dir',
        $map_entry[$key_mtcl], PDO::PARAM_STR);
      $stmt_mtbl->bindValue(':mtbl_mtl_blob',
        $mtl_input, PDO::PARAM_STR);
      $stmt_mtbl->execute();
    }
  }
  catch (PDOException $ex) {
    echo $ex->getMessage();
  }
}
```

Kode 3.6 Pengunggahan pecahan MTL

Kode 3.6 merupakan pseudokode mengunggahan pecahan MTL. *Binding* dengan PDO akan membantu dalam pengunggahan, dengan memastikan pecahan tetap dan tidak bisa diganti saat pengiriman. Binding dibentuk dalam transaksi agar sekali kesalahan dapat membatalkan proses agar kealahan dapat dicari tanpa resiko model yang rusak.

id_blob_mtl	mtbl_id_model	mtbl_mat_name	mtbl_mat_dir	mtbl_mtl_blob
395	29	header_piece	header_piece	["# \r\n","# Wavefront m
396	29	Hgn_BattleCruiser_Main_EXPORT	Hgn_BattleCruiser_Main_EXPORT[1].png	["newmtl Hgn_BattleCruis
397	29	Hgn_BattleCruiserKineticBurstCa	Hgn_BattleCruiserKineticBurstCannon_EXPORT[...	["newmtl Hgn_BattleCruis
398	29	Hgn_BattleCruiserKineticBurstCa0	Hgn_BattleCruiserKineticBurstCannon_EXPORT[...	["newmtl Hgn_BattleCruis
399	29	Hgn_BattleCruiserKineticBurstCa1	Hgn_BattleCruiserKineticBurstCannon_EXPORT[...	["newmtl Hgn_BattleCruis
400	29	Hgn_BattleCruiser_Engine_EXPORT	Hgn_BattleCruiser_Engine_EXPORT[1].png	["newmtl Hgn_BattleCruis
401	29	Hgn_BattleCruiser_Thruster_EXPO	Hgn_BattleCruiser_Thruster_EXPORT[3].png	["newmtl Hgn_BattleCruis
402	29	Hgn_BattleCruiser_Badge_EXPORT	Hgn_BattleCruiser_Badge_EXPORT[1].png	["newmtl Hgn_BattleCruis
403	29	Hgn_BattleCruiser_Main_EXPORT0	Hgn_BattleCruiser_Main_EXPORT[1].png	["newmtl Hgn_BattleCruis
404	29	Hgn_BattleCruiser_Thruster_EXPO0	Hgn_BattleCruiser_Thruster_EXPORT[3].png	["newmtl Hgn_BattleCruis
405	29	Hgn_BattleCruiser_Main_EXPORT1	Hgn_BattleCruiser_Main_EXPORT[1].png	["newmtl Hgn_BattleCruis

Gambar 3.13 Database penyimpanan, tabel blob_MTL

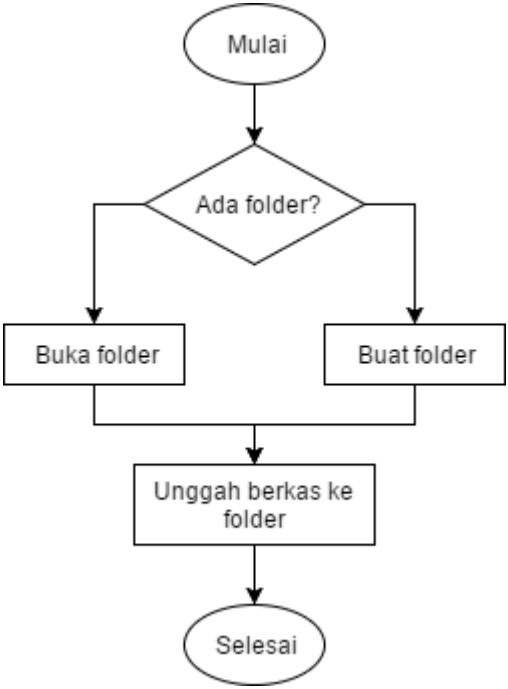
Gambar 3.13 merupakan bentuk tabel penyimpanan pecahan MTL. Pecahan dimasukkan dengan empat data kolom, yaitu ID pecahan, ID model pecahan, nama material pecahan, nama direktori material pecahan, dan pecahan itu sendiri, terkode dalam format JSON. Sistem dua

ID ini akan membantu dalam rekonstruksi kembali model, karena hanya perlu memanggil kembali pecahan yang memiliki kode model tersebut.

Implementasi ini akan mengisi tabel blob_mtl dengan pecahan yang sudah dibagi dan dikonversi serta ident untuk membantu dalam proses pengembalian seperti semula.

3.6. Proses untuk Pecahan Tekstur

Setiap pecahan tekstur akan diproses untuk diunggah ke direktori user, dengan *database* memiliki data dimana pecahan disimpan dan hubungan pecahan dengan entri model. Berikut ini adalah kode untuk unggah pecahan tekstur.



Gambar 3.14 Alur pengunggahan tekstur

Gambar 3.14 merupakan tata alur pengunggahan berkas tekstur. Deteksi adanya folder berkas dilakukan sebelum pengunggahan. Pengunggahan menggunakan sistem try-catch untuk memastikan setiap berkas diterima sempurna.

```
else {
  if ( ! is_dir($path.$username)) {
    mmakeuserdir();
  }
  if ( ! is_dir($path.$username."\\".$modelname)) {
    makemodeldir();
  }

  move_uploaded_file();

  try {
    $stmt_tex->execute(array(
      ':tex_id_model' => $modelId,
      ':tex_name' => $name,
      ':tex_dir' => $upload_path));
  }
  catch (PDOException $ex) {
    echo $ex->getMessage();
  }
}
```

Kode 3.7 Penyimpanan direktori tekstur

Kode 3.7 merupakan pseudokode penyimpanan direktori tekstur. Setiap tekstur akan diberikan entri dalam database berisi ident untuk kelompok model mereka dan direktori dimana mereka berada. Berkas tekstur akan dipindahkan ke dalam direktori pengguna tanpa diberikan nama baru, untuk memfasilitasi penggabungan.

3.7. Restrukturisasi Berkas Model dan Pemanggilan Berkas Tekstur

Pada saat *viewer* meminta model, akan terjadi proses restrukturisasi berkas model dan pemanggilan berkas tekstur. Restrukturisasi akan membuat kembali berkas model OBJ dan MTL dari hasil kompresi yang tersimpan di *database*. Pemanggilan akan menyediakan arah direktori dimana berkas tekstur yang diperlukan

berada. Kedua hal ini yang dibutuhkan untuk dapat membuat *viewer* dapat menampilkan model. Berikut ini adalah kode restrukturisasi dan pemanggilan berkas-berkas.

```
$model_id = get('modelid');
$raw_obj = '';

$stmt_obj = $pdo->prepare(
  "SELECT obj_blob
   FROM blob_obj
   WHERE id_model = ?"
);
$stmt_obj->execute([$model_id]);
$row_obj  = $stmt_obj->fetchAll(PDO::FETCH_COLUMN);

foreach ($row_obj as $proc_obj) {
  foreach (json_decode($proc_obj)
  echo $string_obj;
}
```

Kode 3.8 Restrukturisasi obyek

Kode 3.8 merupakan peseudokode restrukturisasi objek. Dalam kode ini variabel `$dom_obj` berisi berkas utuh OBJ, `$dom_mtl` berisi berkas utuh MTL, dan `$dom_path` berisi arah direktori berkas tekstur yang dibutuhkan. Ketiga variabel ini siap untuk diberikan pada *viewer* untuk ditampilkan pada pengguna.

3.8. Penampilan Model

Penampilan model akan menggunakan WebGL yang dibantu dengan Threejs yang diubah untuk menampilkan model ke layar. Variabel yang sebelumnya dibuat dari restrukturisasi akan diberikan ke *renderer* berikut dengan arah penyimpanan tekstur. Berikut ini adalah kode *renderer* model.

Ada tiga fungsi dasar yang diperlukan dalam penampilan 3D standar menggunakan three.js, yaitu `init()`,`animate()`, dan `render()`. Dari ketiga fungsi ini, `init()` akan menginisiasikan area tampilan untuk memunculkan citra 3D serta memuat berkas-berkas yang diperlukan

untuk menghasilkan citra 3D tersebut. Dalam kasus ini, berkas yang diperlukan adalah berkas OBJ dan MTL yang telah dikonversi menjadi entri data, serta berkas-berkas tekstur yang telah diunggah ke dalam server. Untuk mencapai tahap ini, diperlukan suatu fungsi yang dapat memuat kembali hasil rekonstruksi data OBJ dan MTL yang telah didapatkan seperti dalam bagian 3.7. Fungsi `getter(model_id)` akan menerima ID numerik dari model yang diminta pengguna dan akan mengarahkan fungsi yang ditunjukkan pada Kode 3.8 untuk mengambil data dari *database* dan merekonstruksi kembali menjadi bentuk yang dapat diambil oleh fungsi `init()`. Bentuk dari fungsi adalah sebagai berikut.

Fungsi juga akan memanggil fungsi init dan animate dalam runtime agar dapat langsung meneruskan hasil yang sudah tersimpan dalam variabel ke dalam fungsi init. Berikut adalh fungsi init.

```
var container, stats;
var camera, scene, renderer;
var windowX = $("#viewer").width();
var windowY = windowX * (56.25 / 100);
var windowHalfX = windowX / 2;
var windowHalfY = windowY / 2;
var div_obj, div_mtl;

getter(model_id);

function getter(model_id) {
  var xhr_mtl = new XMLHttpRequest();
  console.time('MTL XHR');
  xhr_mtl.open('GET', '$modelid'+
    model_id, true);
  xhr_mtl.setRequestHeader();
  xhr_mtl.onreadystatechange = function () {
    if (xhr_mtl.readyState == 4) {
      if (xhr_mtl.status == 200) {
        console.timeEnd('MTL XHR');
        div_mtl = xhr_mtl.responseText;

        var xhr_obj = new XMLHttpRequest();
        console.time('OBJ XHR');
        xhr_obj.open('GET', '$modelid' +
          model_id, true);
        xhr_obj.setRequestHeader();
        xhr_obj.onreadystatechange = function () {
```

```
          if (xhr_obj.readyState == 4) {
            if (xhr_obj.status == 200) {
              console.timeEnd('OBJ XHR');
              div_obj = xhr_obj.responseText;
              init();
              animate();
            }
          }
        };
        xhr_obj.send(null);
      }
    }
  };
  xhr_mtl.send(null);
};

function init() {
  container = document.getElementById('viewer');
  aspectRatio = windowX / windowY;

  camera = new THREE.PerspectiveCamera(45, aspectRatio, 1, 10000);
  camera.position.y = 0;

  // scene
  scene = new THREE.Scene();

  var ambient = new THREE.AmbientLight(0xffffff, 1.0);
  var keyLight = new THREE.DirectionalLight(
    new THREE.Color('desired_color'), 1.0);
  keyLight.position.set(-100, 0, 100);
  var fillLight = new THREE.DirectionalLight(
    new THREE.Color('desired_color'), 0.75);
  fillLight.position.set(100, 0, 100);
  var backLight = new THREE.DirectionalLight(0xffffff, 1.0);
  backLight.position.set(100, 0, -100).normalize();

  scene.add(ambient);
  scene.add(keyLight);
  scene.add(fillLight);
  scene.add(backLight);

  var onProgress = function (xhr) {
    if (xhr.lengthComputable) {
      var percentComplete = xhr.loaded / xhr.total * 100;
      console.log( Math.round(
        percentComplete, 2) + '% downloaded' );
    }
  };
```

```
var onError = function (xhr) { };

THREE.Loader.Handlers.add( /\.dds$/i, new THREE.DDSLoader() );

var mtlLoader = new THREE.MTLLoader();
mtlLoader.setPath(div_path);
mtlLoader.load('', div_mtl, function(materials) {
  materials.preload();
  var objLoader = new THREE.OBJLoader();
  objLoader.setMaterials(materials);
  objLoader.setPath(div_path);
  objLoader.load('', div_obj, function (object) {
    var bbox = new THREE.Box3().setFromObject(object);

    console.log(bbox.min);
    console.log(bbox.max);

    bbox.getCenter( object.position );
      // this re-sets the mesh position
    object.position.multiplyScalar( - 1 );

    // camera focus
    var height = bbox.max.y;
    var width = bbox.max.x - bbox.min.x;
    var vertical_FOV = camera.fov * (Math.PI/ 180);
    var horizontal_FOV = 2 * Math.atan (
      Math.tan (vertical_FOV/2) * aspectRatio);

    var max_z = bbox.max.z;

    var distance_vertical = height / (
      2 * Math.tan(vertical_FOV/2));
    var distance_horizontal = width / (
      2 * Math.tan(horizontal_FOV/2));
    var z_distance = distance_vertical >= distance_horizontal?
      distance_vertical : distance_horizontal;
    var cam_distance = z_distance + max_z;

    camera.position.z = cam_distance + (
      cam_distance * (15 / 100));
    camera.position.y = 0 ;
    camera.position.x = 0;

    scene.add(object);
  }, onProgress, onError);
});
```

```
  // renderer
  renderer = new THREE.WebGLRenderer();
  renderer.setPixelRatio(window.devicePixelRatio);
  renderer.setSize(windowX, windowY);
  renderer.setClearColor(new THREE.Color("hsl(0, 0%, 10%)"));
  container.appendChild(renderer.domElement);

  controls = new THREE.OrbitControls(camera, renderer.domElement);
  controls.enableDamping = true;
  controls.dampingFactor = 0.25;
  controls.enableZoom = true;

  window.addEventListener( 'resize', onWindowResize, false );
}

function onWindowResize() {
  windowHalfX = windowX / 2;
  windowHalfY = windowY / 2;

  camera.aspect = windowX / windowY;
  camera.updateProjectionMatrix();

  renderer.setSize(windowX, windowY);
}

function animate() {
  requestAnimationFrame(animate);
  controls.update();
  render();
}

function render() {
  renderer.render(scene, camera);
}
```

Kode 3.9 Rendering obyek

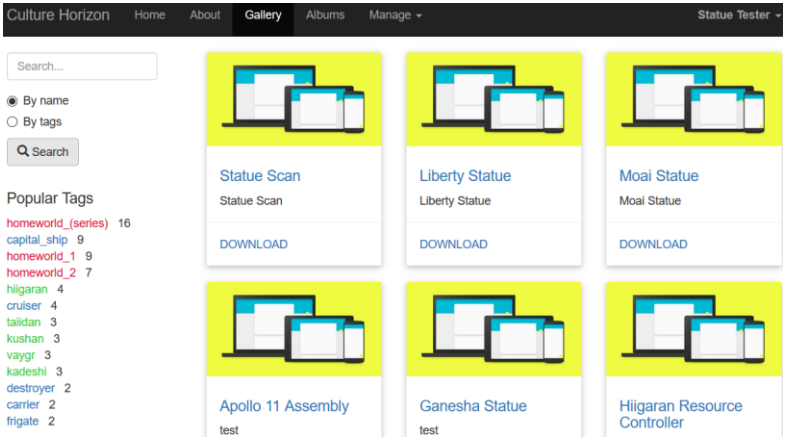
Kode 3.9 merupakan pseudokode rendering untuk objek. Dalam kode ini, akan diset kontainer untuk menyimpan penampilan model. Kontainer akan kemudian diberikan kamera untuk menampilkan model, dan dibuat lingkungan yang memiliki pencahayaan sebagai tempat menampilkan model. Variabel yang sudah dibuat akan diberikan ke *loader* OBJ dan MTL untuk diproses, dengan *loader* MTL memuat material dan tekstur serta *loader* OBJ memuat model. Keduanya

kemudian akan digabungkan dan diberikan ke *renderer* untuk ditampilkan ke lingkungan, serta kontroler untuk kamera. Terakhir, *renderer* akan menampilkan lingkungan berserta kamera ke dalam kontainer.

3.9. Kategorisasi Model

3.9.1. Galeri dan *Tagging* Model

Model akan dikategorikan dalam *database* sesuai dengan *tag* yang didapatkan model dan album untuk model. Sistem *tagging* menggunakan kategorisasi yang terdiri dari *tag* pemilik, hak cipta, karakter, dan ciri-ciri, dengan setiap kategori *tag* memiliki warna tersendiri. Berikut ini adalah contoh galeri penampilan model

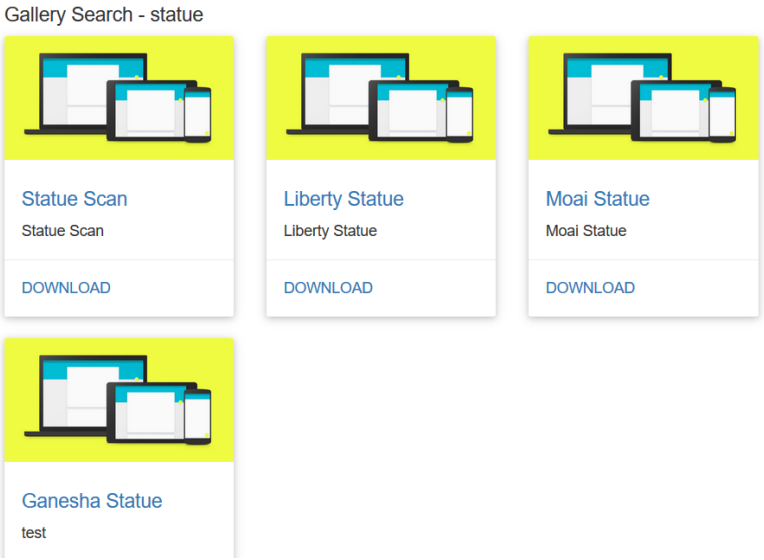


.Gambar 3.15 Contoh galeri model

Gambar 3.15 merupakan tampilan untuk galeri model. Setiap model memiliki *thumbnail* sendiri, beserta dengan nama dan deskripsi singkat model. Galeri memiliki sistem pencarian yang dapat menggunakan nama maupun tagging.

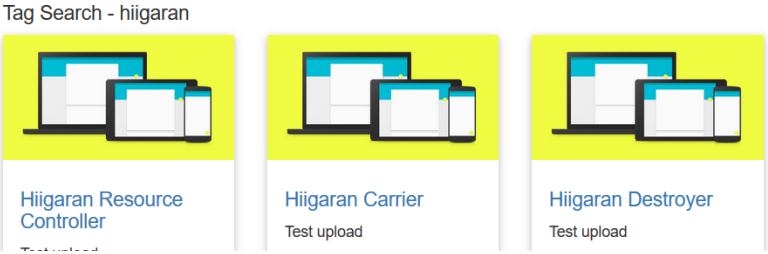
Galeri model memiliki beberapa elemen, yaitu pencarian, *tagging*, dan baris entri model. Pencarian galeri menerima pencarian berdasarkan nama entri model atau berdasarkan *tag* model, dengan hasil pencarian

ditampilkan dalam baris entri. Pencarian akan diurutkan berdasarkan tanggal unggah entri model secara *descending*. Berikut ini adalah contoh penggunaan pencarian.



Gambar 3.16 Contoh penggunaan pencarian berdasarkan nama entri

Gambar 3.16 merupakan contoh pencarian berdasarkan nama entri. Ketika dilakukan pencarian dengan kata sandi *statue*, maka sistem akan menampilkan model-model yang namanya sesuai dengan kata yang dimasukkan dalam pencarian. Pencarian akan mengklasifikasikan berdasarkan waktu model diunggah dalam mengurutkan pencarian, beserta dengan *tagging* mayoritas model-model yang berhasil didapatkan oleh sistem pencari.

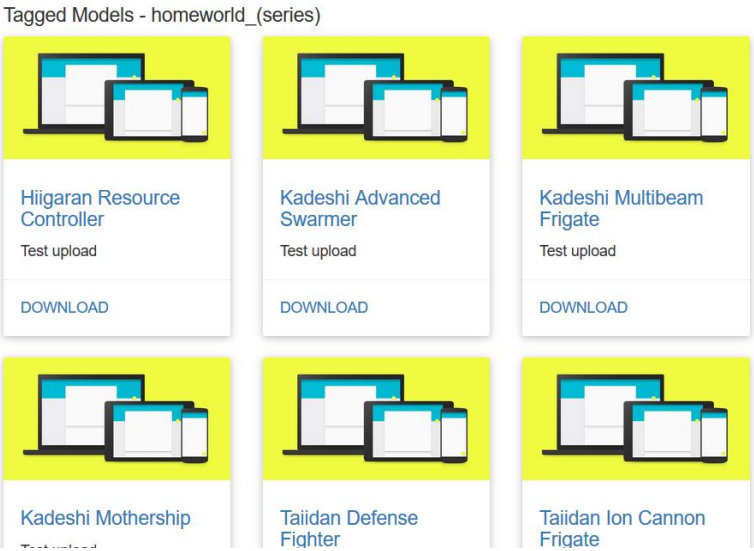


Gambar 3.17 Contoh penggunaan pencarian berdasarkan nama *tag*

Gambar 3.17 merupakan contoh pencarian berdasarkan nama tag. Ketika dilakukan pencarian dengan kata tag *hiigaran*, maka sistem akan menampilkan model-model yang memiliki tag yang sesuai dengan kata yang dimasukkan dalam pencarian. Pencarian akan mengklasifikasikan berdasarkan waktu model diunggah dalam mengurutkan pencarian, beserta dengan *tagging* mayoritas model-model yang berhasil didapatkan oleh sistem pencari.

Sistem *tagging* membantu untuk mencari entri model sesuai dengan kategori yang direpresentasikan oleh setiap *tag*. Contohnya, diinginkan untuk mencari semua kapal jelajah, maka dipilih *tag* yang merepresentasikan kapal jelajah, dan akan ditampilkan semua entri yang memiliki *tag* tersebut.

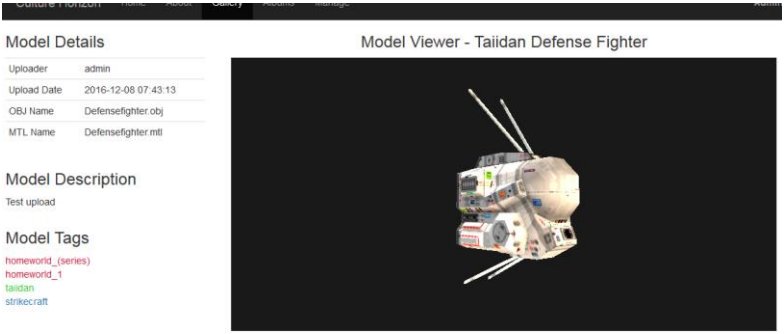
Sistem tag dibagi menjadi beberapa kategori, yang dibedakan dengan warnanya. Tag berwarna merah merupakan tag pemilik, yang dapat berarti hak cipta atau pembuat model. Tag berwarna hijau merupakan tag karakter, seperti aktor dalam model dan sejenisnya. Tag berwarna biru merupakan tag sifat, seperti corak patung Jawa, meriam, dan sebagainya. Setiap tag tidak menggunakan spasi dalam pemisahan kata untuk lebih mudah dalam pencarian. Berikut ini adalah contoh penggunaan *tagging*.



Gambar 3.18 Contoh penggunaan *tagging*

Gambar 3.18 merupakan contoh pencarian berdasarkan tag. Ketika dilakukan pencarian dengan menggunakan tag *homeworld_(series)*, maka sistem akan menampilkan model-model yang memiliki tag yang sesuai dengan tag yang dimasukkan dalam pencarian. Pencarian akan mengklasifikasikan berdasarkan waktu model diunggah dalam mengurutkan pencarian, beserta dengan *tagging* mayoritas model-model yang berhasil didapatkan oleh sistem pencari.

Penampilan model menggunakan WebGL akan meggunakan variabel yang ditentukan diatas untuk membuat *scene* yang dapat ditampilkan pada pengguna. Penampil model dikontrol menggunakan *mouse*, dengan tombol kiri mengontrol perputaran model dan *scrollwheel* mengontrol *zoom* model. Berikut ini adalah contoh penampilan model.



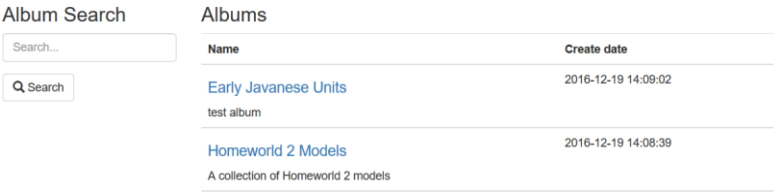
Gambar 3.19 Contoh penampilan model

Gambar 3.19 merupakan contoh penampilan model. Model akan ditampilkan dalam bingkai di sebelah kanan layar, dengan control pergerakan model berada di mouse. Model berada dalam lingkungan gelap agar kontras tekstur model lebih terlihat. Tampilan ini juga menampilkan detil dan hasil listing tag model.

Detil model memberikan nama pengunggah model, deskripsi model, tanggal unggah model, dan nama berkas OBJ dan MTL model. *Listing tag* model memberikan *tag* yang sudah diberikan kepada entri model.

3.9.2. Album Model

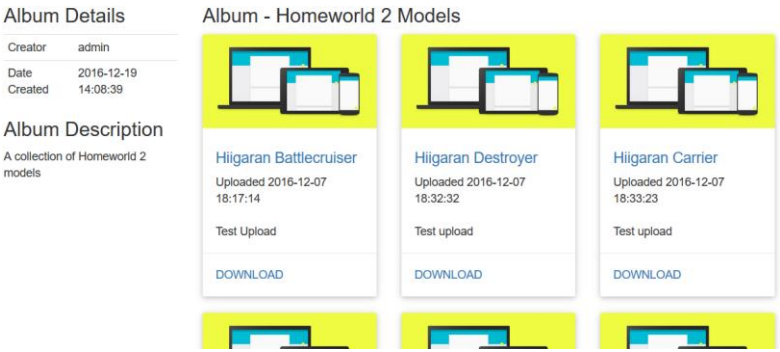
Kategorisasi model juga dapat menggunakan album, dimana model dapat berada dalam satu atau lebih album. Album dapat dibuat dan diubah oleh pengguna yang sudah teregistrasi dalam sistem, dengan album terbuka dapat diubah oleh pengguna siapapun dan album tertutup hanya dapat diubah oleh pengguna yang membuat album itu.. Setiap album memiliki nama dan deskripsi yang membantu dalam pencarian album. Album dapat berisi secara virtual sebanyak mungkin, selama ada yang menambahkan ke dalam album. Entri album juga dapat dicari dengan pencarian berbasis nama entri album. Berikut ini adalah contoh sistem album.



Gambar 3.20 Entri album model

Gambar 3.20 merupakan contoh direktori album. Setiap album memiliki deskripsi sendiri beserta dengan kapan album pertama kali dibuat. Setiap album memiliki entri tersendiri, namun direktori ini merupakan direktori umum, yang membuat semua pengguna dapat melihat dan menggunakan album tersebut, namun tidak semua pengguna dapat mengubah album. Album juga dapat dicari menggunakan sistem pencarian yang terintegrasi.

Setiap album akan memiliki beberapa entri model, yang akan diurutkan dalam album sesuai dengan urutan entri dimasukkan dalam album. Suatu album hanya akan dapat dimasukkan model oleh pemilik album tersebut. Jika album tidak diset sebagai privat, maka album dapat dibuka dan diakses oleh pengguna lain. Berikut ini adalah contoh entri dalam album.

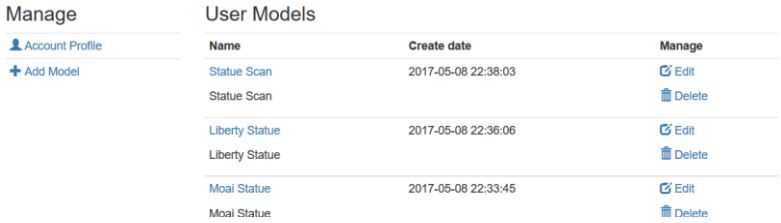


Gambar 3.21 Entri model dalam album

Gambar 3.21 merupakan contoh isi album. Setiap model dalam album akan diurutkan berdasarkan tanggal ungah model, dan setiap model juga memiliki deskripsi model sendiri. Setiap album memiliki tanggal pembuatan album dan deskripsi album sebagai detil mengapa album dibuat.

3.10. Pengaturan Model Unggahan

Pengaturan model unggahan dilakukan pada halaman pengaturan yang muncul ketika pengguna masuk terdaftar ke sistem. Halaman ini merupakan tempat pengaturan untuk akun pengguna, dan juga tempat memasukkan model ke dalam database. Berikut ini adalah tampilan halaman pengaturan.



Gambar 3.22 Halaman pengaturan pengguna

Gambar 3.22 merupakan contoh pengaturan model unggahan. Setiap model memiliki deskripsi model, tanggal buat model, dan sisi penggubahan model. Halaman pengaturan ini juga memiliki profil akun pengguna dan lembar pengunggahan model.

Setiap entri model memiliki tempat pengubahan untuk mengubah seperti *tag*, album, dan lain-lain. Entri model hanya akan bisa diubah atau dihapus jika pengguna pemilik entri masuk dan memutuskan untuk mengubah atau menghapus entri.

Halaman ini sengaja dikosongkan

BAB 4
PENGUJIAN DAN ANALISA

Pada bab ini dibahas mengenai pengujian dari sistem yang telah diimplementasikan untuk mengetahui apakah fungsi sistem yang direncanakan telah bekerja sesuai dengan rancangan.

4.1. Implementasi Perangkat Keras

Pengujian dilakukan dengan komputer dengan sistem operasi Windows 10, dengan browser yang digunakan dalam pengujian adalah Firefox Developer Edition 54.0a2 dan Vivaldi 1.9.818.44. Program yang digunakan dalam pembuatan tugas akhir ini adalah Visual Studio Code untuk menulis dan menganalisa sistem yang dibuat. Untuk Komputer yang digunakan dalam memprogram dan merancang aplikasi ini memiliki spesifikasi sebagai beriku:

Tabel 4.1 Spesifikasi Komputer

Komponen	**Spesifikasi**
Sistem Operasi	Windows 10 Pro 64-bit
Prosesor	Intel Core i7-7500U CPU @ 2,70GHz
Memori	8192MB DDR3 RAM
Versi DirectX	DirectX 11
Graphics Accelerator	NVIDIA GeForce 940MX
GA Dedicated Memory	2048MB DDR3

Tabel 4.1 merupakan spesifikasi komputer yang digunakan dalam pengujian ini, dimana komputer memiliki kartu grafis yang akan membantu dalam *rendering* model 3D. Komputer merupakan jenis unit *laptop* notebook Asus A456U. Dengan spek ini diharapkan komputer tidak terlalu menjadi halangan dalam pengujian.

4.2. Bahan Pengujian

Sistem yang telah dibuat akan diuji dalam kekuatan dalam pengunggahan dan pengunduhan data dari pengguna ke database dan sebaliknya. Dalam hal ini, dibutuhkan bahan berkas model dengan tingkat kompleksitas yang bervariasi untuk dapat menguji sistem dengan beban yang bervariasi. Pengujian juga akan dilakukan untuk menentukan batas pemecahan mana yang paling sesuai dalam menyeimbangkan waktu tunggu pengguna dan ketahanan sistem. Pengujian ini akan menggunakan empat model tertentu, yang masing-masing akan diuji dalam lima batasan pemecahan, yaitu 250kb, 500kb, 750kb, 1000kb, dan 2000kb. Tabel dan gambar detil keempat model uji adalah sebagai berikut.

Tabel 4.2 Berkas yang akan diunggah

Model	**Spesifikasi**
Moai Statue (GameCube - DreamMix TV World Fighters – Moai)	1576 lines, 40.981 bytes
Liberty Statue (wwnnsthl4k-LibertyStatue)	109751 lines, 3.724.356 bytes
Museum Piece #3 (redo-improved-textures-of-museum-piece-3)	534398 lines, 20.402.425 bytes
Patung Ganesha (ganeshatexturemappingfix.obj)	661765 lines, 28.680.239 bytes

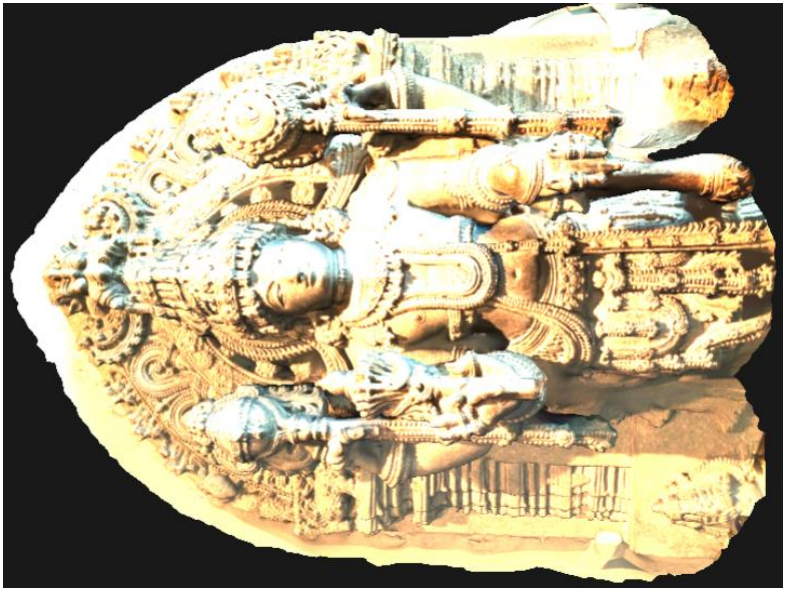
Tabel 4.2 merupakan spesifikasi setiap berkas yang diujikan dalam pengujian ini. Dari keempat berkas, Moai Statue didesignasikan sebagai target lowpoly, Liberty Statue didesignasikan sebagai highpoly, Museum Piece #3 sebagai superhighpoly, dan Patung Ganesha sebagai ultrahighpoly.

Gambar 4.1 Moai Statue

Gambar 4.1 adalah gambar citra Moai Statue yang mewakili objek pengujian low-poly. Objek ini diambil dari *game DreamMix TV World Fighters*. Model memiliki fitur berupa besar berkas 40.981 byte, dan memiliki 1576 baris dalam berkas. Seperti yang terihat di Gambar 4.1, model bersifat low-poly karena sedikitnya polygon yang digunakan, sehingga model terlihat kasar.

Gambar 4.2 Liberty Statue

Gambar 4.2 adalah gambar citra Liberty Statue yang mewakili objek pengujian high-poly. Model memiliki fitur berupa besar berkas 3.724.356 byte, dan memiliki 109.751 baris dalam berkas. Model bersifat high-poly karena banyaknya polygon yang digunakan, sehingga model terlihat lebih halus.



Gambar 4.3 Museum Piece #3

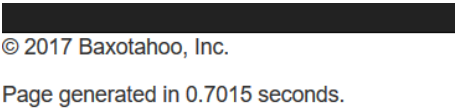
Gambar 4.3 adalah gambar citra Museum Piece #3 yang mewakili objek pengujian superhigh-poly. Objek ini merupakan model hasil pindaian dari sudut tertentu, sehingga model tidak pada mengelilingi dan berlubang di belakang. Model memiliki fitur berupa besar berkas 534.398 lines dan 20.402.425 bytes. Model bersifat high-p-poly karena banyaknya polygon yang digunakan, sehingga model terlihat halus. Selain itu, karena model merupakan hasil pindaian tekstur model memiliki pencahayaan semu yang berada langsung dalam tekstur akibat pencahayaan patung saat pemindaian.

Gambar 4.4 Patung Ganesha

Gambar 4.4 adalah gambar citra Patung Ganesha yang mewakili objek pengujian ultrahigh-poly. Objek ini merupakan model pindaian dengan pindaian 360 derajat, sehingga semua sisi patung terpindai dan menghasilkan model yang padat. Model memiliki fitur berupa besar berkas 661765 lines dan 28.680.239 bytes. Model berupa high-poly karena jumlah polygon yang berada dalam model, dan juga karena detail yang berhasil ditangkap dalam pemindaian patung.

4.3. Cara Pengujian

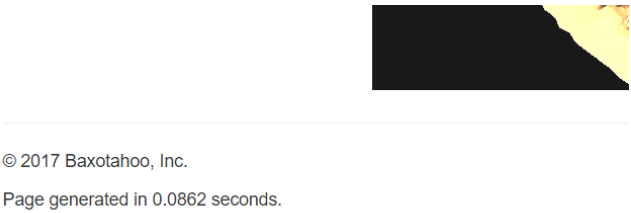
Cara pengujian adalah dengan mengukur seberapa cepat hasil proses algoritma yang digunakan dalam proses pengunggahan dan pengunduhan objek. Pengukuran penggunggahan adalah melalui pengukuran seberapa cepat waktu pemuatan halaman konfirmasi pengunggahan selesai. Berikut ini adalah contoh halaman konfirmasi.



Gambar 4.5 Pengukuran waktu unggah objek

Gambar 4.5 merupakan bagian dari halaman konfirmasi bahwa pengunggahan selesai, yang digunakan sebagai pengukur kecepatan unggah data. Waktu generasi halaman menunjukkan seberapa cepat proses pengunggahan berlangsung karena halaman telah diset untuk ditampilkan setelah semua proses pengunggahan selesai, dan pengukuran waktu diset akan dimulai segera setelah proses pengunggahan berlangsung.

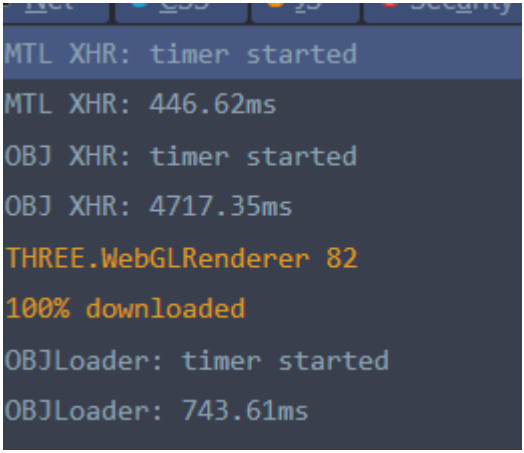
Pengukuran pengunduhan adalah dengan mengukur empat titik waktu, yaitu seberapa cepat pembuatan kembali berkas OBJ, seberapa cepat pembuatan kembali berkas MTL, seberapa cepat OBJLoader bekerja, dan seberapa cepat halaman *viewer* termuat di layar pengguna. Hal ini digunakan karena pemuatan objek bersifat asinkron, dimana pemuatan halaman penampil tidak menunggu pemuatan berkas dari database, sehingga halaman menjadi lebih cepat dalam pemuatannya. Berikut ini adalah contoh pengukuran waktu pemuatan.



Gambar 4.6 Pengukuran waktu muat halaman

Gambar 4.6 merupakan bagian dari halaman penampil model, yang digunakan sebagai pengukuran kecepatan penampilan halaman

penampil model. Penampilan halaman ini akan lebih cepat dibandingkan dengan pengunggahan, karena generasi halaman dan generasi penampil memiliki sifat asinkron, untuk membantu pengguna agar tidak menganggap halaman berhenti bekerja. Hal in penting agar pengalamann penggunaan tetap lancar.



Gambar 4.7 Pengukuran waktu muat objek

Gambar 4.7 merupakan bagian dari konsol browser, yang memiliki bagian lain dari pengukuran kecepatan pengunduhan. Pengukuran di wilayah ini memiliki tiga tahap, yaitu pemanggilan MTL, pemanggilan OBJ, dan pemuatan data ke loader. Pewaktuan di tahap ini akan bersifat sekuensial, dengan tahap dari atas ke bawah. Pemuatan biasanya akan memiliki *bottleneck* di bagian OBJ, karena besarnya data yang harus diambil dari database.

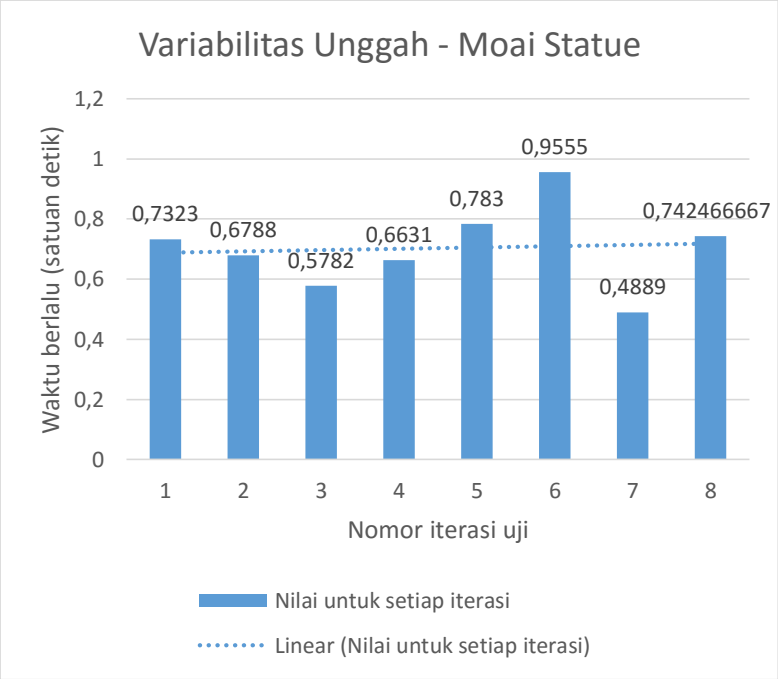
Data yang didapatkan akan berupa data iterasi pengujian untuk setiap taha digabungkan dalam bentuk tabel, yang akan diambil rata-ratanya dan disajikan dalam bentuk grafik dengan bagian sesuai dengan data yang didapatkan.

4.4. Pengujian Pengunggahan ke Database

Pengujian ini akan mengukur seberapa cepat sistem dapat memproses berkas yang diberikan, sesuai dengan besar berkas dan batas pemecahan yang ditetapkan admin. Parameter ukuran dalah seberapa cepat halaman konfirmasi pengunggahan muncul, diukur dalam detik

4.4.1. Moai Statue

Model memiliki ukuran yang kecil, dan karena itu masuk di bawah batas pemecahan apapun. Oleh karena itu, model hanya akan dipecah sesuai dengan batasan objek dalam berkas. Berikut ini adalah variabilitas pengunggahan objek ini.



Grafik 4.1 Variabilitas pengunggahan *Moai Statue*

Grafik 4.1 adalah visualisasi variabilitas dari pengunggahan Moai Statue. Sumbu Y mewakili waktu yang dibutuhkan, dan sumbu X mewakili nomor dari setiap iterasi pengujian, dengan dilakukan 8 kali iterasi pengujian. Variasi pada waktu pemuatan merupakan variasi acak yang terjadi pada pengukuran iterasi yang berasal dari perangkat pengujian, dengan nilai tertinggi berupa 0,9555 detik dan nilai terendah berupa 0,5782 detik, dan rata-rata waktu unggahan adalah 0,7028 setik, diambil dari 8 kali pengujian.

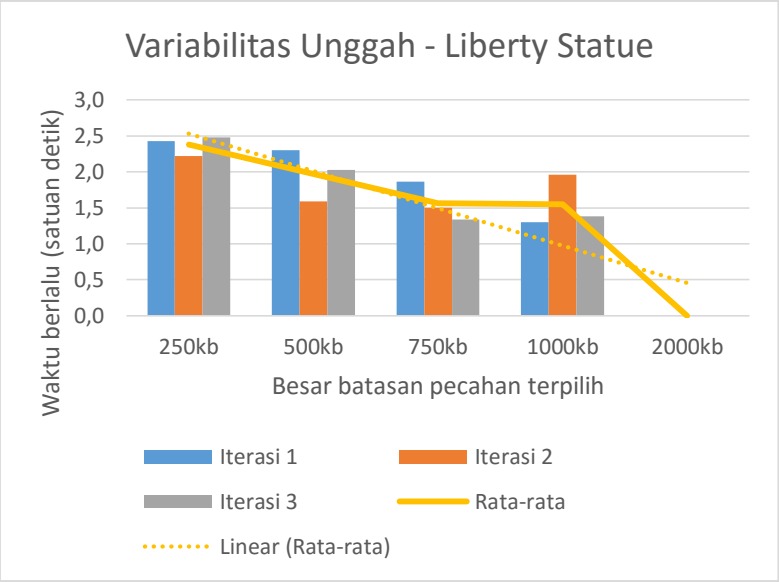
4.4.2. Liberty Statue

Model memiliki ukuran yang melebihi batas pemecahan, maka dilakukan pemecahan berdasarkan batas pemecahan. Ukuran model yang tergolong sedang mempercepat proses pegnunggahan, dan semakin besar batas pengunggahan, terlihat pemuatan semakin cepat. Pada batasan 2000kb, batasan telah melebihi batas transaksi MySQL, yang berarti model tidak dapat diunggah. Berikut ini adalah tabel dan grafik variabilitas unggahan untuk Liberty Statue.

Tabel 4.3 Pengujian unggahan untuk Liberty Statue.

Besar pecahan	Iterasi 1 (s)	Iterasi 2 (s)	Iterasi 3 (s)	Rata-rata (s)
250kb	2,43000	2,22330	2,48200	2,37843
500kb	2,30400	1,59270	2,02830	1,97500
750kb	1,86430	1,50410	1,33440	1,56760
1000kb	1,30260	1,95860	1,38070	1,54730
2000kb	0,00000	0,00000	0,00000	0,00000

Tabel 4.3 merupakan hasil pengujian untuk Liberty Statue. Pengujian berhasil dilakukan hingga batasan 2000kb, dimana sistem tidak bisa menerima data karena melebihi besar data yang bisa diproses oleh sistem setiap saat. Pengujian menggunakan 3 iterasi umum, dimana setiap besar batasan yang dipilih akan diuji kecepatannya berulang tiga kali yang kemudian akan dibandingkan dan diambil rata-ratanya.



Grafik 4.2 Variabilitas pengunggahan *Liberty Statue*

Grafik 4.2 adalah visualisasi variabilitas dari pengunggahan Liberty Statue. Sumbu Y mewakili waktu yang dibutuhkan, dan sumbu X mewakili nomor dari setiap iterasi pengujian, dengan dilakukan 3 kali iterasi pengujian untuk setiap batasan pecahan data. Hasil yang tidak sama pada setiap iterasi dari besar batasan yang sama merupakan variasi acak dari perangkat yang digunakan.

Terlihat dari Grafik 4.2 bahwa variasi pada waktu pemuatan cenderung semakin menurun seiring dengan besarnya batasan yang ditetapkan, namun pada tingkat 2000kb sistem tidak bisa menerima data karena keterbatasan dari sistem *backend* dan menggagalkan pengunggahan.

4.4.3. Museum Piece #3

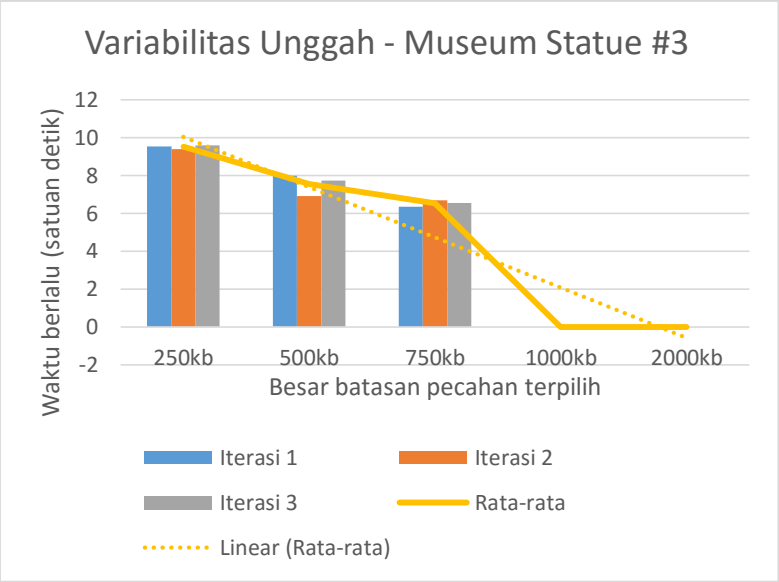
Model memiliki ukuran yang melebihi batas pemecahan, dan karena itu model melalui pemecahan berdasarkan batas pemecahan. Ukuran model yang besar memperlambat proses pengunggahan, dan

semakin besar batas pengunggahan, terlihat pemuatan semakin cepat. Pada batasan 1000kb, ada hasil pemecahan yang telah melebihi batas transaksi MySQL, yang membua model tidak dapat diunggah. Berikut ini adalah tabel dan grafik variabilitas unggahan untuk Museum Piece #3.

Tabel 4.4 Pengujian unggahan untuk Museum Piece #3.

Besar pecahan	Iterasi 1 (s)	Iterasi 2 (s)	Iterasi 3 (s)	Rata-rata (s)
250kb	9,5573	9,4049	9,6098	9,524
500kb	7,9935	6,9077	7,7402	7,547133
750kb	6,3547	6,7048	6,5624	6,540633
1000kb	0	0	0	0
2000kb	0	0	0	0

Tabel 4.4 merupakan hasil pengujian untuk Museum Piece #3. Pengujian berhasil dilakukan hingga batasan 1000kb, dimana sistem pemisahan mengeluarkan pecahan data yang melebihi batasan yang ditetapkan sehingga sistem *database* tidak dapat menerima data karena melebihi besar data yang bisa diproses oleh sistem setiap saat. Pengujian menggunakan 3 iterasi umum, dimana setiap besar batasan yang dipilih akan diuji kecepatannya berulang tiga kali yang kemudian akan dibandingkan dan diambil rata-ratanya.



Grafik 4.3 Variabilitas pengunggahan *Museum Piece #3*

Grafik 4.3 adalah visualisasi variabilitas dari pengunggahan Museum Statue #3. Sumbu Y mewakili waktu yang dibutuhkan, dan sumbu X mewakili nomor dari setiap iterasi pengujian, dengan dilakukan 3 kali iterasi pengujian untuk setiap batasan pecahan data. Hasil yang tidak sama pada setiap iterasi dari besar batasan yang sama merupakan variasi acak dari perangkat yang digunakan.

Terlihat dari Grafik 4.3 bahwa variasi pada waktu pemuatan cenderung semakin menurun seiring dengan besarnya batasan yang ditetapkan, namun pada tingkat 1000kb, sistem pemecahan mengeluarkan pecahan yang tidak dapat diproses ke dalam sistem *database*, sehingga sistem keseluruhan mengeluarkan *error*. Pada tingkat batasan terpilih 2000kb, sistem tidak bisa menerima data karena keterbatasan dari sistem *backend* dan menggagalkan pengunggahan

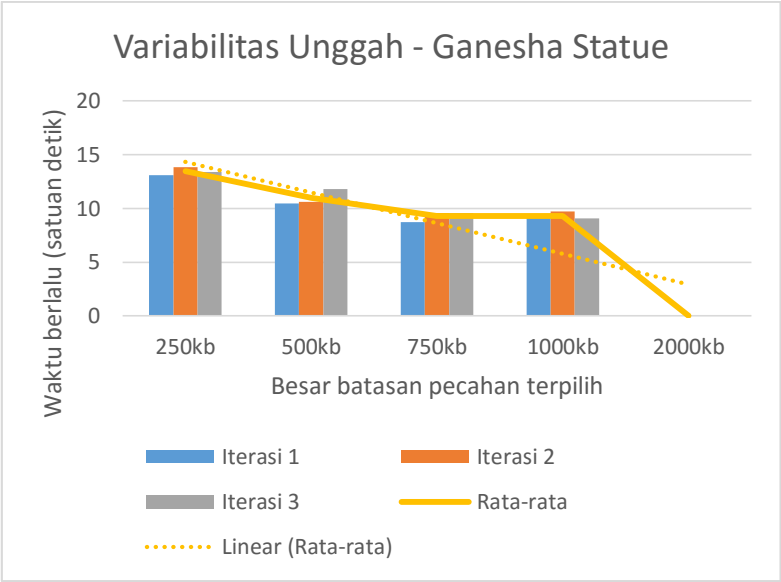
4.4.4. Patung Ganesha

Model memiliki ukuran yang melebihi batas pemecahan, maka dilakukan pemecahan berdasarkan batas pemecahan. Ukuran model yang tergolong sangat besar sangat memperlambat proses penunggahan, dan semakin besar batas pengunggahan, terlihat pemuatan semakin cepat. Pada batasan 2000kb, pemecahan batasan telah melebihi batas transaksi MySQL, yang berarti model tidak dapat diunggah. Berikut ini adalah tabel dan grafik variabilitas unggahan untuk Patung Ganesha.

Tabel 4.5 Pengujian unggahan untuk Patung Ganesha

Besar pecahan	Iterasi 1 (s)	Iterasi 2 (s)	Iterasi 3 (s)	Rata-rata (s)
250kb	13,1046	13,8279	13,4079	13,4468
500kb	10,4805	10,603	11,8259	10,9698
750kb	8,7189	9,5604	9,5891	9,289467
1000kb	9,1434	9,7328	9,0923	9,322833
2000kb	0	0	0	0

Tabel 4.5 merupakan hasil pengujian untuk Patung Ganesha. Pengujian berhasil dilakukan hingga batasan 2000kb, dimana sistem tidak bisa menerima data karena melebihi besar data yang bisa diproses oleh sistem setiap saat. Pengujian menggunakan 3 iterasi umum, dimana setiap besar batasan yang dipilih akan diuji kecepatannya berulang tiga kali yang kemudian akan dibandingkan dan diambil rata-ratanya.



Grafik 4.4 Variabilitas pengunggahan *Patung Ganesha*

Grafik 4.4 adalah visualisasi variabilitas dari pengunggahan Ganesha Statue. Sumbu Y mewakili waktu yang dibutuhkan, dan sumbu X mewakili nomor dari setiap iterasi pengujian, dengan dilakukan 3 kali iterasi pengujian untuk setiap batasan pecahan data. Hasil yang tidak sama pada setiap iterasi dari besar batasan yang sama merupakan variasi acak dari perangkat yang digunakan.

Terlihat dari Grafik 4.2 bahwa variasi pada waktu pemuatan semakin menurun seiring dengan besarnya batasan yang ditetapkan, namun pada tingkat 2000kb sistem tidak bisa menerima data dan menggagalkan pengunggahan. Selain itu, ada kecenderungan efek pembesaran batas pemecahan akan semakin besar seiring dengan semakin besarnya model yang diunggah.

4.5. Pengujian Pengunduhan ke Viewer

Pengujian ini akan mengukur seberapa cepat sistem dapat memproses berkas yangberada di database hingga dapat kembali ditampilkan sebagai citra 3D ke pengguna. Pengukuran memiliki 4 tahap, yaitu saat proses pengumpulan berkas OBJ, pengumpulan berkas MTL, ukuran kecepatan OBJLoader dari three.js, dan kecepatan pembuatan halaman viewer, diukur dalam detik.

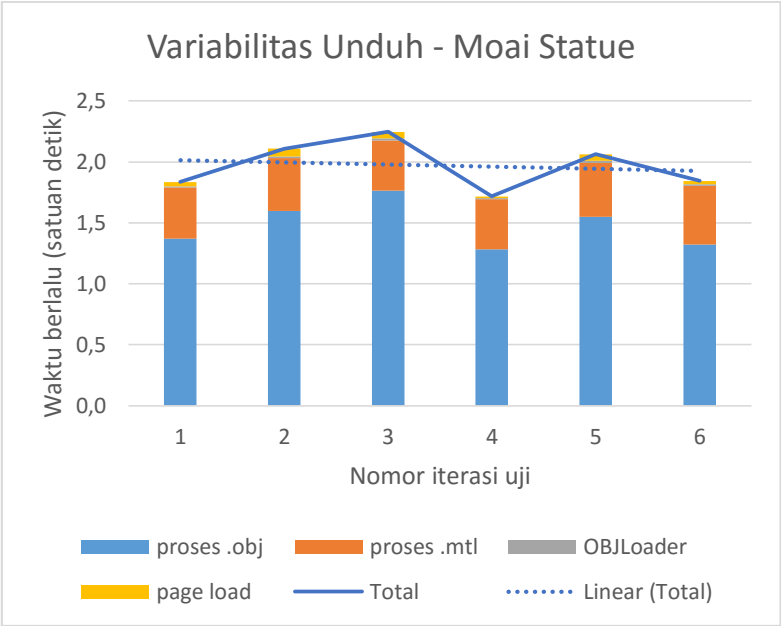
4.5.1. Moai Statue

Model memiliki ukuran yang kecil, dan karena itu masuk di bawah batas pemecahan apapun. Oleh karena itu, model hanya akan dipecah sesuai dengan batasan objek dalam berkas. Berikut ini adalah variabilitas pengunduhan objek ini.

Tabel 4.6 Pengukuran loadtime Moai Statue

No	proses .obj	proses .mtl	OBJ Loader	page load	Total
1	1,36936	0,42088	0,01145	0,03500	1,83669
2	1,59745	0,43529	0,01177	0,06390	2,10841
3	1,76279	0,41526	0,01322	0,05510	2,24637
4	1,28537	0,40928	0,01037	0,01170	1,71672
5	1,55174	0,44203	0,01603	0,05280	2,06260
6	1,32142	0,48399	0,01099	0,02830	1,84470
Avg.	1,48136	0,43446	0,01231	0,04113	1,96925

Tabel 4.6 merupakan hasil pengujian untuk Moai Statue. Variasi dalam pengukuran berasal dari variasi acak yang didapatkan dari variasi beban dle perangkat pengujian. Pengujian memiliki variasi yang cukup standar, dengan fase yang membutuhkan waktu paling banyak adalah fase pengambilan berkas OBJ, yang ditandai dalam kolom **proses .obj**. Rata-rata dari setiap iterasi memiliki kecenderungan mendekati 1,96 detik, dengan variasi rata-rata iterasi mengikuti variasi hasil pengukuran iterasi tersebut.



Grafik 4.5 Variabilitas pengunduhan *Moai Statue*

Grafik 4.5 merupakan visualisasi variabilitas dari pengunduhan Moai Statue. Sumbu Y mewakili waktu yang dibutuhkan, dan sumbu X mewakili nomor dari setiap iterasi pengujian, dengan dilakukan 6 kali iterasi pengujian. Hasil pengujian ditampilkan dalam format *stacked* column, dimana hasil pengukuran proses .obj, proses .mtl, pemuatan OBJ, dan pemuatan halaman ditumpuk untuk membuat kolom total waktu yang diperlukan untuk setiap iterasi. Variasi rata-rata waktu unggahan adalah 1,96925 detik, diambil dari 8 kali pengujian. Rasio proses berkas OBJ yang dilambangkan dengan warna biru muda terhadap proses lain cukup besar dibandingkan dengan proses lain.

4.5.2. Liberty Statue

Model memiliki ukuran yang melebihi batas pemecahan, maka dilakukan pemecahan berdasarkan batas pemecahan. Ukuran model yang tergolong sedang mempercepat proses pengunduhan, dan semakin besar

batas pengunggahan, terlihat pemuatan semakin cepat. Pada batasan 2000kb, batasan telah melebihi batas transaksi MySQL, sehingga model tidak dapat diunggah. Berikut ini adalah tabel dan grafik variabilitas unggahan untuk Liberty Statue.

Tabel 4.7 Pengukuran loadtime Liberty Statue, 250kb

No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	1,90629	0,48970	0,16499	0,05920	2,62018
2	1,88852	0,49531	0,16692	0,03470	2,58545
3	2,13782	0,42772	0,17410	0,03310	2,77274
Avg.	1,97754	0,47091	0,16867	0,04233	2,65946

Tabel 4.7 Pengukuran loadtime Liberty Statue, 500kb

No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	1,80010	0,43945	0,14976	0,03120	2,42051
2	1,90079	0,42687	0,18109	0,05660	2,56535
3	1,80830	0,43556	0,16272	0,06140	2,46798
Avg.	1,83640	0,43396	0,16452	0,04973	2,48461

Tabel 4.8 Pengukuran loadtime Liberty Statue, 750kb

No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	1,66588	0,42013	0,13621	0,06320	2,28542
2	1,66614	0,43631	0,18109	0,05640	2,33994
3	2,12629	0,52652	0,14292	0,10800	2,90373
Avg.	1,81944	0,46099	0,15341	0,07587	2,50970

Tabel 4.9 Pengukuran loadtime Liberty Statue, 1000kb

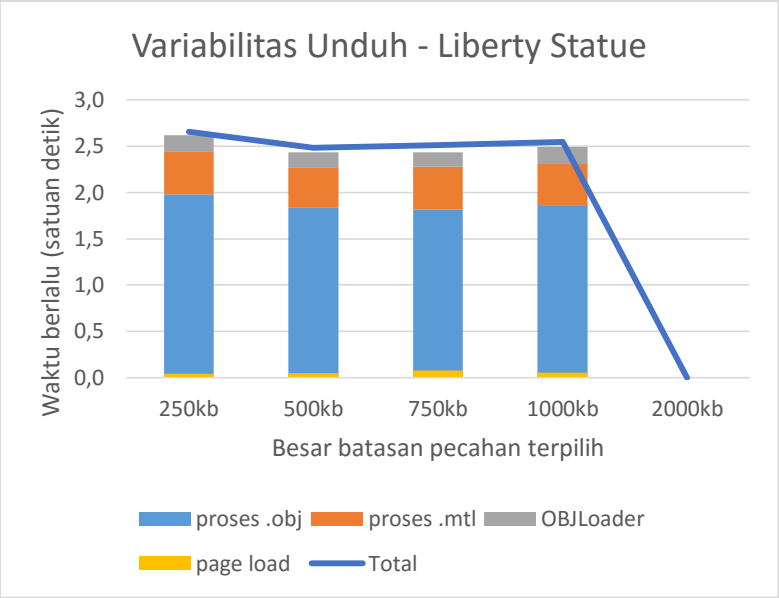
No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	1,77079	0,46965	0,18663	0,06180	2,48887
2	1,88130	0,42413	0,22158	0,05320	2,58021
3	1,94778	0,42337	0,14701	0,05320	2,57136
Avg.	1,86662	0,43905	0,18507	0,05607	2,54681

Tabel 4.6 hingga 4.9 merupakan hasil pengujian Liberty Statue, dengan tabel 4.6 mewakili iterasi untuk batasan 250kb, tabel 4.7 mewakili iterasi untuk batasan 500kb, tabel 4.8 mewakili iterasi untuk batasan 750kb, dan tabel 4.9 mewakili iterasi untuk batasan 1000kb. Setiap tahap pengujian memiliki variasi antar iterasi sesama pilihan batasan yang berupa variasi acak. Variansi antar pilihan batasan cenderung memiliki waktu lebih rendah seiring dengan besar batasan.

Tabel 4.10 Pengukuran loadtime Liberty Statue (rata-rata)

Besar pecahan	proses .obj	proses .mtl	OBJ Loader	page load	Total
250kb	1,97754	0,47091	0,16867	0,04233	2,65946
500kb	1,83640	0,43396	0,16452	0,04973	2,48461
750kb	1,81944	0,46099	0,15341	0,07587	2,50970
1000kb	1,86662	0,43905	0,18507	0,05607	2,54681
2000kb	0,00000	0,00000	0,00000	0,00000	0,00000

Tabel 4.10 merupakan hasil rata-rata pengujian Liberty Statue, dikumpulkan dari hasil rata-rata iterasi setiap pilihan batasan. Rasio untuk pemrosesan data OBJ dari *database*, dilambangkan dengan kolom **proses .obj** terlihat mulai lebih besar dibandingkan dengan rasio proses lain, dengan rasio pemuatan halaman yang semakin kecil. Rasio untuk 2000kb tidak dapat diambil karena tidak dapat diambil pengujian secara sempurna.



Grafik 4.6 Variabilitas pengunduhan *Liberty Statue*

Grafik 4.6 adalah visualisasi variabilitas dari pengunduhan Liberty Statue. Sumbu Y mewakili waktu yang dibutuhkan, dan sumbu X mewakili batasan pecahan yang dipilih. Hasil pengujian ditampilkan dalam format *stacked* column, dimana hasil pengukuran proses .obj, proses .mtl, pemuatan OBJ, dan pemuatan halaman ditumpuk untuk membuat kolom total waktu yang diperlukan untuk setiap iterasi. Variasi rata-rata waktu cenderung semakin turun seiring dengan besar batasan, meskipun terjadi variasi acak yang membuat data menjadi diluar prediksi, dengan batasan terakhir tidak memiliki nilai karena kegagalan. Rasio proses berkas OBJ terhadap proses lain cukup besar dibandingkan dengan proses lain.

4.5.3. Museum Piece #3

Model memiliki ukuran yang melebihi batas pemecahan, maka dilakukan pemecahan berdasarkan batas pemecahan. Ukuran model yang tergolong besar memperlambat proses pengunduhan, dan semakin besar

batas pecahan untuk pengunggahan, terlihat pemuatan semakin cepat. Pada batasan 1000kb, sistem pemecahan mengeluarkan hasil pemecahan batasan yang di luar ambang batas pemecahan dan telah melebihi batas transaksi MySQL, yang membuat model tidak dapat diunggah. Berikut ini adalah tabel dan grafik variabilitas unggahan untuk Museum Piece #3.

Tabel 4.11 Pengukuran loadtime Museum Piece #3, 250kb

No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	4,00583	0,42402	0,82916	0,05650	5,31551
2	4,01481	0,58909	0,88790	0,05760	5,54940
3	4,48025	0,43836	0,84793	0,03060	5,79714
Avg.	4,16696	0,48382	0,85500	0,04823	5,55402

Tabel 4.12 Pengukuran loadtime Museum Piece #3, 500kb

No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	3,04848	0,43745	0,63830	0,05930	4,18353
2	3,41893	0,52556	0,70587	0,02690	4,67726
3	3,21547	0,53617	0,61431	0,06220	4,42815
Avg.	3,22763	0,49973	0,65283	0,04947	4,42965

Tabel 4.13 Pengukuran loadtime Museum Statue #3, 750kb

No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	3,33456	0,42271	0,72846	0,02730	4,51303
2	3,53478	0,47222	0,64785	0,04180	4,69665
3	3,28536	0,42919	0,76007	0,07070	4,54532
Avg.	3,38490	0,44137	0,71213	0,04660	4,58500

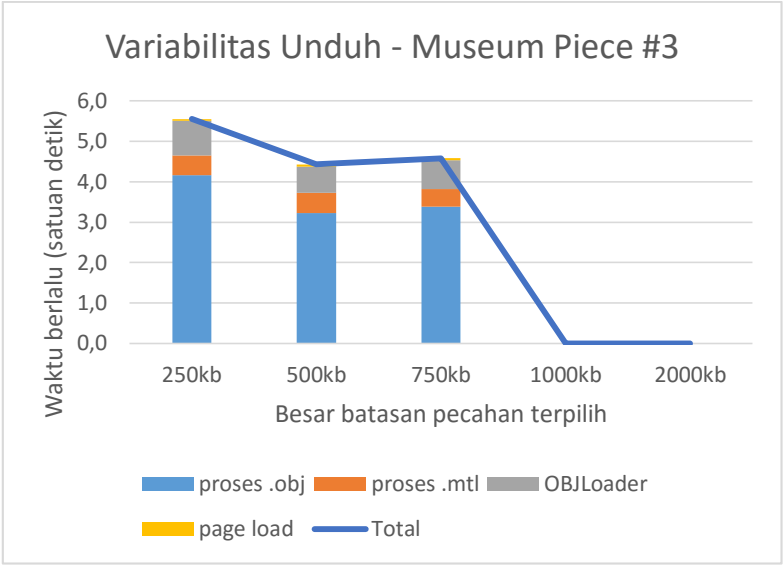
Tabel 4.11 hingga 4.13 merupakan hasil pengujian Museum Statue #3, dengan tabel 4.11 mewakili iterasi untuk batasan 250kb, tabel 4.12 mewakili iterasi untuk batasan 500kb, dan tabel 4.13 mewakili iterasi untuk batasan 750kb. Setiap tahap pengujian memiliki variasi antar iterasi sesama pilihan batasan yang berupa variasi acak. Variansi antar pilihan

batasan cenderung memiliki waktu lebih rendah seiring dengan besar batasan. Terlihat dari ketiga tabel bahwa proses rekonstruksi berkas, ditandai dengan kolom **proses .obj**, dan proses pemuatan berkas ke penampil, ditandai dengan kolom **OBJLoader**, merupakan proses yang paling memakan waktu dalam pemuatan total berkas dari mulai hingga selesai.

Tabel 4.14 Pengukuran loadtime Museum Piece #3 (rata-rata)

Besar pecahan	proses .obj	proses .mtl	OBJ Loader	page load	Total
250kb	4,16696	0,48382	0,85500	0,04823	5,55402
500kb	3,22763	0,49973	0,65283	0,04947	4,42965
750kb	3,38490	0,44137	0,71213	0,04660	4,58500
1000kb	0,00000	0,00000	0,00000	0,00000	0,00000
2000kb	0,00000	0,00000	0,00000	0,00000	0,00000

Tabel 4.14 merupakan hasil rata-rata pengujian Museum Piece #3, dikumpulkan dari hasil rata-rata iterasi setiap pilihan batasan. Rasio proses OBJ, dilambangkan dalam kolom **proses .obj**, terlihat masih lebih besar dibandingkan dengan rasio proses lain, dengan rasio pemuatan halaman yang semakin kecil. Rasio untuk 1000kb dan 2000kb tidak dapat diambil karena data tidak dapat diambil karena tidak dapat diunggah oleh sistem pengunggah.



Grafik 4.7 Variabilitas pengunduhan *Museum Piece #3*

Grafik 4.7 adalah visualisasi variabilitas dari pengunduhan Museum Piece #3. Sumbu Y mewakili waktu, dan sumbu X mewakili nomor test. Hasil pengujian ditampilkan dalam format *stacked* column, dimana hasil pengukuran proses .obj, proses .mtl, pemuatan OBJ, dan pemuatan halaman ditumpuk untuk membuat kolom total waktu yang diperlukan untuk setiap iterasi. Variasi rata-rata waktu semakin turun seiring dengan besar batasan, dengan batasan terakhir tidak memiliki nilai karena kegagalan. Rasio proses berkas OBJ terhadap proses lain cukup besar dibandingkan dengan proses lain.

4.5.4. Patung Ganesha

Model memiliki ukuran yang melebihi batas pemecahan, maka dilakukan pemecahan berdasarkan batas pemecahan. Ukuran model yang sangat besar juga sangat memperlambat proses pengunduhan, dan semakin besar batas pengunggahan, terlihat pemuatan semakin cepat. Pada batasan 2000kb, pemecahan batasan telah melebihi batas transaksi

MySQL, yang berarti model tidak dapat diunggah. Berikut ini adalah tabel dan grafik variabilitas unggahan untuk Patung Ganesha.

Tabel 4.15 Pengukuran loadtime Patung Ganesha, 250kb

No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	5,41442	0,41641	0,93516	0,05640	6,82239
2	5,71936	0,42725	0,90434	0,05960	7,11055
3	5,37696	0,42491	1,03664	0,06610	6,90461
Avg.	5,50358	0,42286	0,95871	0,06070	6,94585

Tabel 4.16 Pengukuran loadtime Patung Ganesha, 500kb

No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	5,41442	0,41641	0,93516	0,05640	6,82239
2	5,71936	0,42725	0,90434	0,05960	7,11055
3	5,37696	0,42491	1,03664	0,06610	6,90461
Avg.	5,50358	0,42286	0,95871	0,06070	6,94585

Tabel 4.17 Pengukuran loadtime Patung Ganesha, 750kb

No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	5,41442	0,41641	0,93516	0,05640	6,82239
2	5,71936	0,42725	0,90434	0,05960	7,11055
3	5,37696	0,42491	1,03664	0,06610	6,90461
Avg.	5,50358	0,42286	0,95871	0,06070	6,94585

Tabel 4.18 Pengukuran loadtime Patung Ganesha, 1000kb

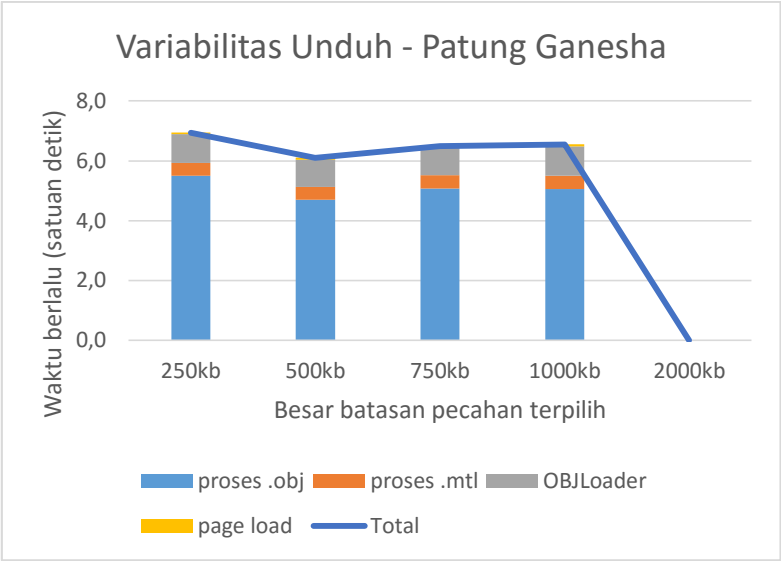
No	proses .obj	proses .mtl	OBJ Loader	page load	Total (detik)
1	5,41442	0,41641	0,93516	0,05640	6,82239
2	5,71936	0,42725	0,90434	0,05960	7,11055
3	5,37696	0,42491	1,03664	0,06610	6,90461
Avg.	5,50358	0,42286	0,95871	0,06070	6,94585

Tabel 4.15 hingga 4.18 merupakan hasil pengujian Patung Ganesha, dengan tabel 4.15 mewakili iterasi untuk batasan 250kb, tabel 4.16 mewakili iterasi untuk batasan 500kb, tabel 4.17 mewakili iterasi untuk batasan 750kb, dan tabel 4.18 mewakili iterasi untuk batasan 1000kb. Setiap tahap pengujian memiliki variasi antar iterasi sesama pilihan batasan yang berupa variasi acak. Terlihat dari keempat tabel bahwa proses rekonstruksi berkas, ditandai dengan kolom **proses .obj**, dan proses pemuatan berkas ke penampil, ditandai dengan kolom **OBJLoader**, merupakan proses yang paling memakan waktu dalam pemuatan total berkas dari mulai hingga selesai.

Tabel 4.19 Pengukuran loadtime Patung Ganesha, rata-rata

Besar pecahan	proses .obj	proses .mtl	OBJ Loader	page load	Total
250kb	5,50358	0,42286	0,95871	0,06070	6,94585
500kb	4,69028	0,43975	0,91326	0,05943	6,10272
750kb	5,07941	0,44509	0,90923	0,05313	6,48686
1000kb	5,06208	0,44676	0,97802	0,06100	6,54786
2000kb	0,00000	0,00000	0,00000	0,00000	0,00000

Tabel 4.19 merupakan hasil rata-rata pengujian Patung Ganesha. Rasio proses OBJ, dilambangkan dalam kolom **proses .obj**, terlihat masih lebih besar dibandingkan dengan rasio proses lain, dengan rasio pemuatan halaman yang semakin kecil..Rasio untuk 2000kb tidak dapat diambil karena tidak dapat diambil pengujian secara sempurna.



Grafik 4.8 Variabilitas pengunduhan *Patung Ganesha*

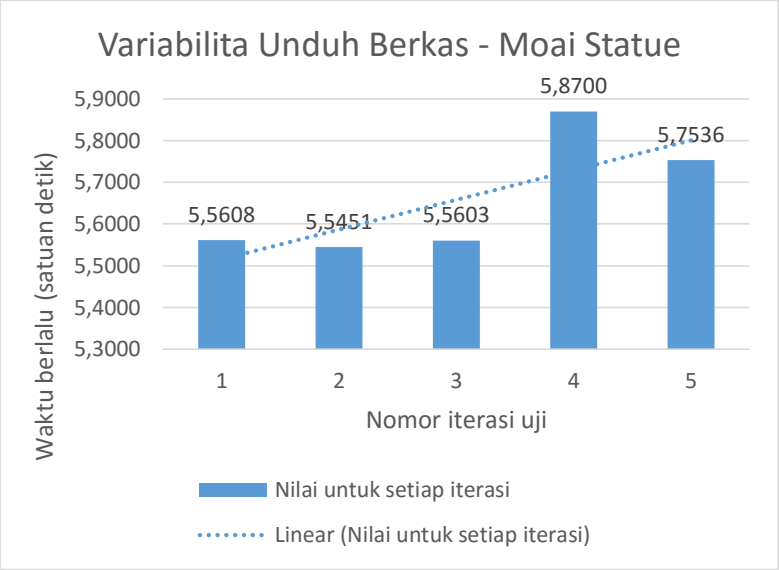
Grafik 4.8 adalah visualisasi variabilitas dari pengunduhan Patung Ganesha. Sumbu Y mewakili waktu, dan sumbu X mewakili nomor test. Hasil pengujian ditampilkan dalam format *stacked* column, dimana hasil pengukuran proses .obj, proses .mtl, pemuatan OBJ, dan pemuatan halaman ditumpuk untuk membuat kolom total waktu yang diperlukan untuk setiap iterasi. Variasi rata-rata waktu semakin turun seiring dengan besar batasan, dengan batasan terakhir tidak memiliki nilai karena kegagalan. Rasio proses berkas OBJ terhadap proses lain terlihat sangat besar dibandingkan dengan proses lain.

4.6. Pengujian Unduh ke Berkas

Pengujian ini akan mengukur seberapa cepat sistem dapat memproses datai dari *database* yang diberikan dan membuatnya menjadi berkas utuh yang dapat dibuka oleh aplikasi *offline*, sesuai dengan besar berkas dan batas pemecahan yang ditetapkan admin. Parameter ukuran dalah seberapa cepat halaman konfirmasi pengunggahan muncul, diukur dalam detik

4.6.1. Moai Statue

Model memiliki ukuran yang kecil, dan karena itu masuk di bawah batas pemecahan apapun. Oleh karena itu, model hanya akan dipecah sesuai dengan batasan objek dalam berkas. Berikut ini adalah variabilitas pengunduhan berkas objek ini.



Grafik 4.9 Variabilitas pengunduhan berkas *Moai Statue*

Grafik 4.9 adalah visualisasi variabilitas dari pengunduhan berkas Moai Statue. Sumbu Y mewakili waktu yang dibutuhkan, dan sumbu X mewakili nomor dari setiap iterasi pengujian, dengan dilakukan 5 kali iterasi pengujian. Variasi pada waktu pemuatan menjadi berkas merupakan variasi acak yang terjadi pada pengukuran iterasi yang berasal dari perangkat pengujian, dengan nilai tertinggi berupa 5,8700 detik dan nilai terendah berupa 5,7536 detik, dan rata-rata waktu unggahan adalah 0,7028 detik, diambil dari 5 kali pengujian.

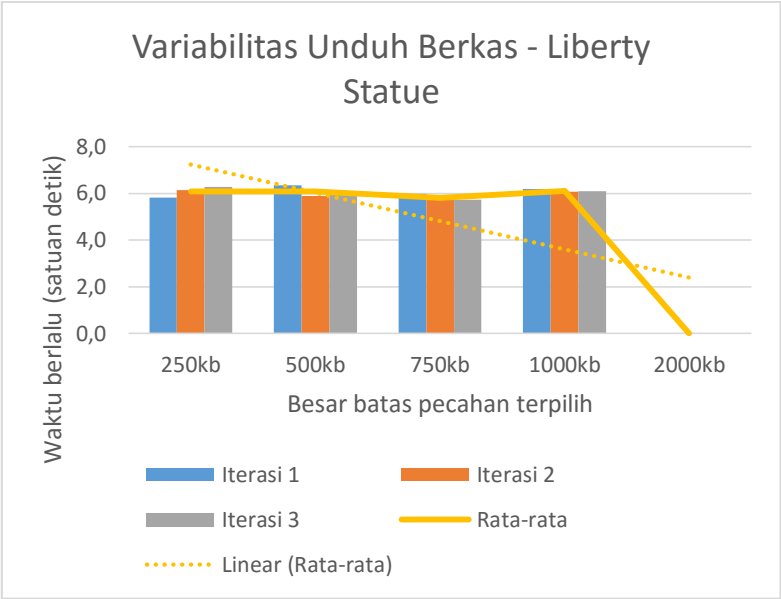
4.6.2. Liberty Statue

Model memiliki ukuran yang melebihi batas pemecahan, maka dilakukan pemecahan berdasarkan batas pemecahan. Ukuran model yang tergolong sedang mempercepat proses pengunduhan, dan semakin besar batas pengunggahan, ada kecenderungan terlihat pengunduhan semakin cepat, namun juga ada kemungkinan disebabkan oleh variasi acak. Pada batasan 2000kb, batasan telah melebihi batas transaksi MySQL, yang berarti model tidak dapat diunggah. Berikut ini adalah tabel dan grafik variabilitas pengunduhan berkas untuk Liberty Statue.

Tabel 4.20 Pengujian unduhan berkas untuk Liberty Statue.

Besar pecahan	Iterasi 1 (s)	Iterasi 2 (s)	Iterasi 3 (s)	Rata-rata (s)
250kb	5,81467	6,14182	6,27997	6,07882
500kb	6,33388	5,89569	6,00880	6,07946
750kb	5,98110	5,72724	5,73501	5,81445
1000kb	6,17486	6,07495	6,08155	6,11045
2000kb	0,00000	0,00000	0,00000	0,00000

Tabel 4.20 merupakan hasil pengujian pengunduhan berkas untuk Liberty Statue. Pengujian berhasil dilakukan hingga batasan 2000kb, dimana sistem tidak bisa menerima data pada saat pengunggahan karena melebihi besar data yang bisa diproses oleh sistem setiap saat. Pengujian menggunakan 3 iterasi umum, dimana setiap besar batasan yang dipilih akan diuji kecepatannya berulang tiga kali yang kemudian akan dibandingkan dan diambil rata-ratanya.



Grafik 4.10 Variabilitas pengunduhan berkas *Liberty Statue*

Grafik 4.10 adalah visualisasi variabilitas dari pengunduhan berkas Liberty Statue. Sumbu Y mewakili waktu yang dibutuhkan, dan sumbu X mewakili nomor dari setiap iterasi pengujian, dengan dilakukan 3 kali iterasi pengujian untuk setiap batasan pecahan data. Hasil yang tidak sama pada setiap iterasi dari besar batasan yang sama merupakan variasi acak dari perangkat yang digunakan.

Terlihat dari Grafik 4.10 bahwa variasi pada waktu pemuatan tidak terlalu dipengaruhi oleh batas pecahan yang ditetapkan, namun pada tingkat 2000kb sistem tidak bisa menerima data karena keterbatasan dari sistem *backend* dan menggagalkan pengunggahan sebelum pengujian.

4.6.3. Museum Piece #3

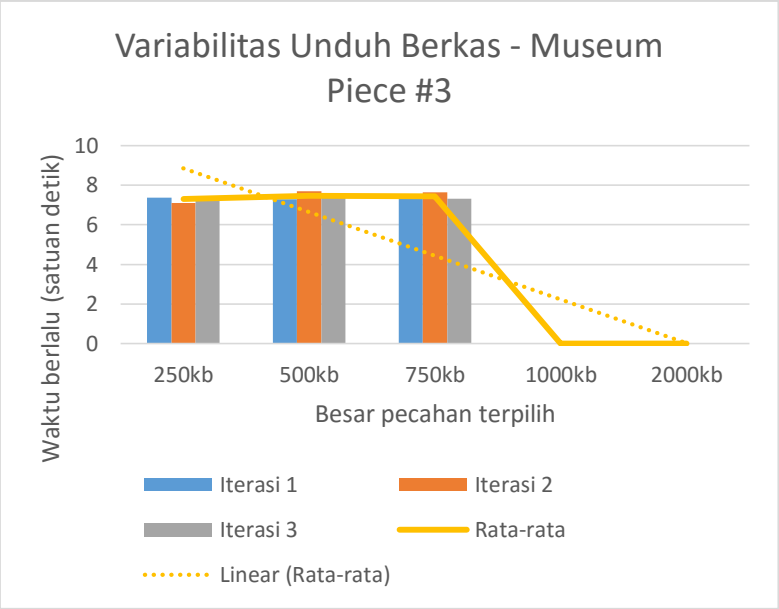
Model memiliki ukuran yang melebihi batas pemecahan, dan karena itu model melalui pemecahan berdasarkan batas pemecahan. Ukuran model yang besar memperlambat proses pengunduhan berkas,

dan semakin besar batas pecahan yang dipilih, terlihat kemungkinan proses pengunduhan berkas semakin cepat, namun juga ada kemungkinan terjadi variasi acak. Pada batasan 1000kb, ada hasil pemecahan yang telah melebihi batas transaksi MySQL, yang membua model tidak dapat diunggah. Berikut ini adalah tabel dan grafik variabilitas unduhan berkas untuk Museum Piece #3.

Tabel 4.21 Pengujian unduhan berkas untuk Museum Piece #3.

Besar pecahan	Iterasi 1 (s)	Iterasi 2 (s)	Iterasi 3 (s)	Rata-rata (s)
250kb	7,37664	7,11505	7,43566	7,309117
500kb	7,28794	7,6842	7,45159	7,474577
750kb	7,35336	7,6286	7,31281	7,43159
1000kb	0	0	0	0
2000kb	0	0	0	0

Tabel 4.21 merupakan hasil pengujian pengunduhan berkas untuk Museum Piece #3. Pengujian berhasil dilakukan hingga batasan 1000kb, dimana sistem pemisahan mengeluarkan pecahan data yang melebihi batasan yang ditetapkan sehingga sistem *database* tidak dapat menerima data karena melebihi besar data yang bisa diproses oleh sistem setiap saat. Pengujian menggunakan 3 iterasi umum, dimana setiap besar batasan yang dipilih akan diuji kecepatannya berulang tiga kali yang kemudian akan dibandingkan dan diambil rata-ratanya.



Grafik 4.11 Variabilitas pengunduhan berkas *Museum Piece #3*

Grafik 4.11 adalah visualisasi variabilitas dari pengunggahan Museum Piece #3. Sumbu Y mewakili waktu yang dibutuhkan, dan sumbu X mewakili nomor dari setiap iterasi pengujian, dengan dilakukan 3 kali iterasi pengujian untuk setiap batasan pecahan data. Hasil yang tidak sama pada setiap iterasi dari besar batasan yang sama merupakan variasi acak dari perangkat yang digunakan.

Terlihat dari Grafik 4.11 bahwa variasi acak pada waktu pengunduhan cenderung membuat waktu yang dibutuhkan menjadi lebih besar seiring dengan besarnya batasan yang ditetapkan, namun pada tingkat 1000kb, sistem pemecahan mengeluarkan pecahan yang tidak dapat diproses ke dalam sistem *database*, sehingga sistem keseluruhan mengeluarkan *error*. Pada tingkat batasan terpilih 2000kb, sistem tidak bisa menerima data karena keterbatasan dari sistem *backend* dan menggagalkan pengunggahan

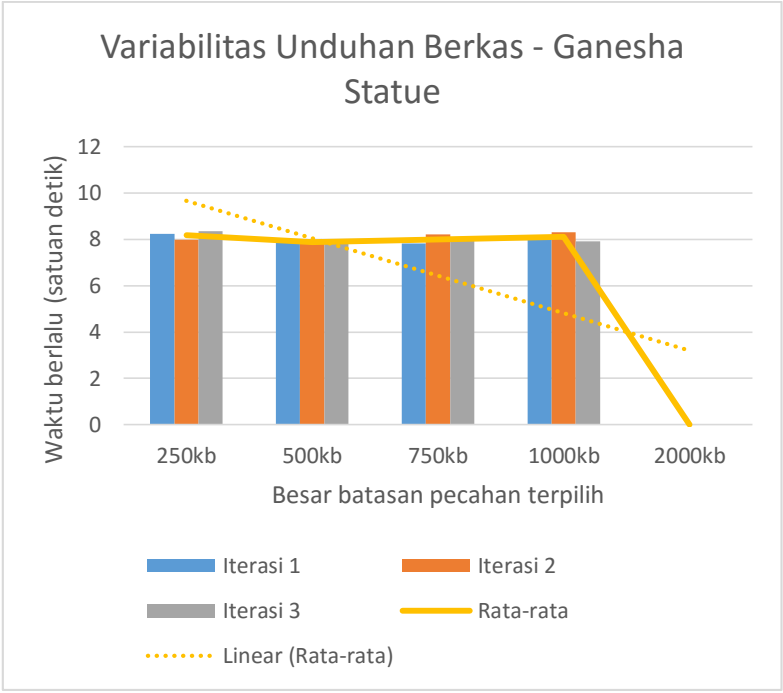
4.6.4. Patung Ganesha

Model memiliki ukuran yang melebihi batas pemecahan, maka dilakukan pemecahan berdasarkan batas pemecahan. Ukuran model yang tergolong sangat besar mempercepat proses penunggahan, dan semakin besar batas pengunggahan, terlihat pemuatan semakin cepat. Pada batasan 2000kb, pemecahan batasan telah melebihi batas transaksi MySQL, yang berarti model tidak dapat diunggah. Berikut ini adalah tabel dan grafik variabilitas unggahan untuk Patung Ganesha.

Tabel 4.22 Pengujian pengunduhan berkas untuk Patung Ganesha

Besar pecahan	Iterasi 1 (s)	Iterasi 2 (s)	Iterasi 3 (s)	Rata-rata (s)
250kb	8,2251	7,9835	8,3408	8,183133
500kb	7,9215	7,8704	7,857	7,882967
750kb	7,8107	8,2108	7,9351	7,985533
1000kb	8,12004	8,3075	7,9173	8,114947
2000kb	0	0	0	0

Tabel 4.22 merupakan hasil pengujian pengunduhan berkas untuk Patung Ganesha. Pengujian berhasil dilakukan hingga batasan 2000kb, dimana sistem tidak bisa menerima data karena melebihi besar data yang bisa diproses oleh sistem setiap saat. Pengujian menggunakan 3 iterasi umum, dimana setiap besar batasan yang dipilih akan diuji kecepatannya berulang tiga kali yang kemudian akan dibandingkan dan diambil rata-ratanya.



Grafik 4.12 Variabilitas pengungduhan berkas *Patung Ganesha*

Grafik 4.12 adalah visualisasi variabilitas dari pengunduhan berkas Ganesha Statue. Sumbu Y mewakili waktu yang dibutuhkan, dan sumbu X mewakili nomor dari setiap iterasi pengujian, dengan dilakukan 3 kali iterasi pengujian untuk setiap batasan pecahan data. Hasil yang tidak sama pada setiap iterasi dari besar batasan yang sama merupakan variasi acak dari perangkat yang digunakan.

Terlihat dari Grafik 4.12 bahwa variasi acak pada waktu pengunduhan membuat waktu tempuh tidak terlalu berubah seiring dengan perubahan batas pecahan, namun pada tingkat 2000kb sistem tidak bisa menerima data dan menggagalkan pengunggahan. Variasi acak lebih memengaruhi perubahan waktu tempuh dibandingkan perubahan batas pecahan dalam pengujian model yang sama.

Halaman ini sengaja dikosongkan

BAB 5
PENUTUP

5.1. Kesimpulan

Berdasarkan pada hasil perancangan, simulasi dan pengujian dari keseluruhan sistem dalam Tugas Akhir ini, maka diperoleh kesimpulan sebagai berikut.

1. Besar berkas model akan mempengaruhi bagaimana berkas OBJ/MTL akan diproses.
2. Besar pemecahan berkas berpengaruh terhadap proses dalam sistem, dimana semakin besar batas pecahan yang dipilih akan berpengaruh pengunggahan dan pengunduhan berkas menjadi lebih cepat, namun hanya dalam batas tertentu dan tidak dapat terlalu besar, sesai dengan kelemahan dari MySQL.
3. Penampilan asinkron akan membantu untuk menghindari pengguna terkesan aplikasi telah mengalami *hang* dengan menampilkan halaman lebih cepat sebelum hasil pemuatan selesai muncul

5.2. Saran

Berdasarkan pengujian dan kesimpulan yang telah didapat, muncul beberapa kritik dan saran yang dapat diperhatikan untuk pengembangan aplikasi ini di masa yang akan datang. Beberapa kritik dan saran tersebut diantaranya adalah sebagai berikut.

1. Pencacahan dan restrukturisasi dapat diadaptasi untuk hanya menampilkan bagian model yang diinginkan, selama berkas model mendukungnya.
2. Integrasi pemisahan dan pencacahan data secara asinkron untuk membantu mempercepat pemuatan data ke dalam *database.*

Apabila penelitian ini akan dikembangkan lebih lanjut pada masa yang akan datang, disarankan untuk menggunakan sumber daya yang

sekurang-kurangnya sama atau lebih besar dari yang digunakan pada penelitian ini.

DAFTAR PUSTAKA

[1] **Jiménez, Jesús, Cruz, Jaime, and Ruiz de Miras, Juan,** "*High performance 3D visualization on the Web: a biomedical case study*," Department of Computer Science, University of Jaén. 2013.

[2] **Bahor, Senad,** "*HTML5/WebGL vs Flash in 3D Visualisation*," Sarajevo School of Science and Technology. 2013.

[3] **Holmberg, Nathan, Wünsche, Burkhard, and Tempero, Ewan,** "*A Framework for Interactive Web-Based Visualization*," 2008.

[4] **Zudilova-Seinstra, Elena, & Jomier, Julien,** "*Bringing 3D visualization to online research articles*," Elsevier. Retrieved 2016. https://www.elsevier.com/connect/bringing-3d-visualization-to-online-research-articles.

[5] **Amirebrahimi, Sam, and Rajabifard, Abbas,** "*An Integrated Web-Based 3d Modeling and Visualization Platform to Support Sustainable Cities*," Center for SDIs and Land Administration (CSDILA), Department of Infrastructure Engineering, The University of Melbourne. 2012.

Halaman ini sengaja dikosongkan

BIODATA

Lintang Anugrah dilahirkan pada tanggal 13 Februari 1995 di kota Malang. Penulis bersekolah di TK Tadika Puri Medan, SDN 11 Manado, SMPN 19 Jakarta, dan SMA Labschool Kebayoran sebelum melanjutkan pendidikan lanjut di Institut Teknologi Sepuluh Nopember Surabaya. Prestasi tertinggi penulis dalam kegiatan adalah sebagai finalis Olimpiade Matematika SD Tingkat Kabupaten di Manado.

Halaman ini sengaja dikosongkan