IMPLEMENTASI *2D SKELETAL BASED ANIMATION ENGINE* MENGGUNAKAN *OPENGL*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Mokhamad Zukhruf Mifta Al Firdaus
NIM: 155150207111015

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

PENGESAHAN

IMPLEMENTASI *2D SKELETAL BASED ANIMATION ENGINE* MENGGUNAKAN *OPENGL*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Mokhamad Zukhruf Mifta Al Firdaus
NIM: 155150207111015

Skripsi ini telah diuji dan dinyatakan lulus pada
17 Juli 2019
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Eriq M. Adams Jonemaro, S.T, M.Kom	Tri Afirianto, S.T, M.T
NIP: 19850410 201212 1 001	NIK: 201309 851213 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 17 Juni 2019

M. Zukhruf Mifta Al Firdaus

NIM: 155150207111015

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa yang telah melimpahkan rahmat, taufik dan hidayah-Nya sehingga skripsi dengan judul "Implementasi 2D *Skeletal Based Animation Engine* Menggunakan *OpenGL*" dapat terselesaikan. Penulis menyadari bahwa skripsi ini tidak akan berhasil tanpa bantuan dari beberapa pihak. Oleh karena itu, penulis ingin menyampaikan rasa hormat dan terima kasih kepada:

1. Allah SWT yang dengan Hidayah serta Kuasa-Nya, penulis tidak akan mampu menyelesaikan skripsi.
2. Orang Tua penulis yang dengan segala usaha, doa, dan tirakatnya agar penulis dapat menyelesaikan skripsi dengan baik.
3. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer.
4. Bapak Agus Wahyu Widodo, S.T, M.Cs selaku Ketua Program Studi Teknik Informatika Fakultas Ilmu Komputer.
5. Bapak Eriq Muhammad Adams Jonemaro, S.T, M.Kom selaku dosen pembimbing satu Skripsi.
6. Bapak Tri Afirianto, S.T, M.T selaku dosen pembimbing dua Skripsi.
7. Nabila Lubna Irbakanisa, S.Kom yang telah memberikan dukungan penuh serta doa kepada penulis agar dapat menyelesaikan Skripsi.
8. Habridio Kurniawan Putra, Ade Wija Nugraha, Vicky Firdaus, Muhammad Faris Faishal, Fathi Nur Azzam, Ade Arta, Hanif Baihaqi, Agung Wicaksono, dan teman teman alumni kelas B Maba yang tetap solid hingga saat ini.
9. Kepada teman-teman E-Sport Arcana yang menemani penulis bermain disaat merasa jenuh, Dhimas, Awin, Fajar, Mbak Fel, Anthea, Diva, Angga, Gusti, Om Nasrul, serta Om Agus.
10. Teman teman PK2 Maba 2017 dan PK2 Maba 2018 yang telah memberikan pengalaman yang sangat berharga dan inspirasi kepada penulis.

Penulis menyadari bahwa dalam penyusunan skripsi masih banyak kekurangan, sehingga saran dan kritik yang membangun sangat penulis harapkan. Akhir kata penulis berharap skripsi ini dapat memberikan manfaat bagi semua pihak yang menggunakannya.

Malang, 11 April 2019

Penulis

zukhruf.m.f@gmail.com

ABSTRAK

Mokhamad Zukhruf Mifta Al Firdaus, Implementasi *2D Skeletal Based Animation Engine* Menggunakan *OpenGL*

Pembimbing: Eriq Muhammad Adams Jonemaro, S.T, M.Kom dan Tri Afirianto, S.T, M.T

Dalam pembuatan animasi 2D, terdapat beberapa teknik yang cukup populer salah satunya adalah teknik *skeletal-based animation*. Teknik *skeletal-based animation* merupakan teknik animasi, dimana animator hanya membutuhkan potongan bagian dari karakter, kemudian disesuaikan letaknya sesuai dengan kerangka maya, teknik ini sama dengan teknik animasi pada 3D. Namun dalam perkembangannya, teknologi seperti *motion capture* maupun 3D *laser scanner* dalam pembuatan animasi 3D belum digunakan dalam pembuatan animasi 2D, terutama dengan teknik *skeletal-based animation*. Dan untuk menghasilkan animasi 2D dibutuhkan 50 hingga 300 animator untuk menggambar ribuan karakter dalam pose yang berbeda dengan menggunakan tangan, hal tersebut menyebabkan proses pembuatan animasi 2D menjadi melelahkan karena sebagian prosesnya dilakukan secara manual serta memakan waktu. Penilitian ini berfokus kepada implementasi *2D skeletal-based animation engine* menggunakan *OpenGL*. *Animation engine* yang diimplementasikan menggunakan API *OpenGL* karena *OpenGL* merupakan API yang khusus digunakan untuk pengolahan grafis, baik 2D maupun 3D. Serta menggunakan sensor *Kinect* yang merupakan salah satu teknologi *motion capture* yang tidak memerlukan marker. Hasil dari penelitian menunjukkan hasil valid pada pengujian fungsionalitas untuk komponen *AnimateFrame* serta *playImportKeyframe*, hal ini menunjukkan bahwa *animation engine* berhasil diimplementasikan dengan menggunakan API OpenGL serta sensor *Kinect* dan dapat menghasilkan animasi dengan teknik *skeletal-based animation*.

Kata kunci: *Skeletal-based Animation*, *Animation Engine, OpenGL, Kinect*

ABSTRACT

Mokhamad Zukhruf Mifta Al Firdaus, *2D Skeletal Based Animation Engine Implementation Using OpenGL*

***Adviser*: Eriq Muhammad Adams Jonemaro, S.T, M.Kom and Tri Afirianto, S.T, M.T**

In making 2D animation there are several techniques which are quite popular, one of them is skeletal-based animation technique. Skeletal-based animation is a technique where animator only needs parts form character, then adjust the location according to a virtual skeleton, this technique is the same as the animation technique in 3D. But in its development, motion capture or 3D laser scanner technology in making 3D animation have not been used for making 2D animation, especially with skeletal-based animation. And to produce 2D animation needs 50 to 300 animator to draw thousands of characters in different poses with human hands. This causes the animator mostly laborious and time-consuming in making animation. This research focuses on the implementation of 2D skeletal-based animation engine using OpenGL. Animation engine that is implemented using the OpenGL API because OpenGL is a specific API for graphics processing, both 2D and 3D. Also using a Kinect sensor which is one of motion capture technology that no need marker. The result of this research shows the valid result on functionality testing for AnimateFrame and playImportKeyframe component, it shows the animation engine successfully implemented by using OpenGL API and Kinect sensor and can produce animation with skeletal-based animation technique.

Keywords: *Skeletal-based Animation, Animation Engine, OpenGL, Kinect*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Dalam pembuatan animasi 2 dimensi (2D), terdapat beberapa teknik yang cukup populer digunakan oleh animator dan salah satunya adalah teknik *skeletal-based animation*. Pada teknik *skeletal-based animation*, animator hanya membutuhkan satu gambar karakter yang telah di potong-potong sesuai dengan bagian tubuhnya kemudian menambahkan *skeleton virtual* yang sesuai dengan karakter untuk membuat satu animasi (Lehtonen, 2016). Teknik ini memiliki kemiripan dengan teknik animasi pada animasi 3 dimensi (3D), yaitu menggunakan *skeleton* untuk menggerakan karakter (Dai et al., 2010).

Dalam pembuatan animasi 3D, teknologi-teknologi yang digunakan telah sangat maju seperti teknologi *motion capture* dan *3D laser scanner*. Namun, dengan perkembangan animasi 3D yang telah menggunakan teknologi tersebut dalam pembuatannya, animasi 2D belum diuntungkan dengan pencapaian tersebut dan untuk menghasilkan animasi 2D, dibutuhkan 50 hingga 300 animator untuk menggambar ribuan karakter dalam pose yang berbeda dengan menggunakan tangan. Hal tersebut menyebabkan proses pembuatan animasi 2D menjadi melelahkan karena sebagian prosesnya dilakukan secara manual serta memakan waktu. Penelitian yang dilakukan oleh Pan dan Zhang pada tahun 2011 dengan judul "*Skeleton-based skeleton driven 2D animation and motion capture*" menghasilkan perangkat lunak yang menerapkan konsep animasi 3D *skeleton-driven* pada *sketch-based animation 2D*. Perangkat lunak itu bertujuan untuk memudahkan animator dalam membuat animasi 2D dengan menerapkan konsep *skeleton-driven animation* pada *sketch-based animation* (Pan dan Zhang, 2011).

Kinect merupakan salah satu teknologi *motion capture* yang mampu menerima input alami dari manusia. Banyak penelitian yang menggunakan teknologi ini untuk berbagai kepentingan. Penelitian dengan judul "*Human Motion Tracking & Evaluation using Kinect V2 Sensor*" menggunakan teknologi Kinect sebagai masukan untuk aplikasi rehabilitasi medis dan latihan olahraga untuk para trainee ketika pelatih tidak ada di tempat untuk secara langsung melatih para trainee (Alabbasi et al., 2015). Adapun juga penelitian dengan judul "*Basic dance pose applications using Kinect technology*" yang menghasilkan aplikasi pembelajaran tari tradisional yang bertujuan untuk mengenalkan tari tradisional, khususnya tari remo, sekaligus menjaga kelestariannya (Ramadijanti, Fahrul and Pangestu, 2016).

Menurut Dewen dan Hongxia (2009), menggunakan *library OpenGL* untuk menghasilkan aplikasi komputer grafis secara *real-time* sangat efisien karena *OpenGL* menyediakan fungsi-fungsi yang mendukung untuk komputer grafis yang umum digunakan untuk *virtual-reality*, animasi komputer, dan *visualisasi*. Dengan adanya masalah seperti yang telah dijelaskan, penggunaan teknologi *Kinect* serta *OpenGL* akan berguna jika digunakan dalam bidang animasi 2D dengan teknik *skeletal-based animation*. Maka dari itu penelitian ini menawarkan sebuah solusi yaitu menggunakan teknologi *Kinect* dalam pembuatan animasi 2D dengan teknik

skeletal-based animation dengan tujuan agar animator dapat dengan mudah membuat animasi 2D dengan teknik *skeletal-based animation*.

1.2 Rumusan Masalah

Bedasarkan penjelasan latar belakang, penulis merumuskan rumusan masalah yaitu:

1. Bagaimana arsitektur pada *2D skeletal based animation engine*?
2. Bagaimana hasil pengujian fungsionalitas dari 2D *skeletal based animation engine*?

1.3 Tujuan

Adapun tujuan yang ingin dicapai dari penelitian ini yaitu mengimplementasikan *2D skeletal based animation engine* dengan menggunakan library *OpenGL*.

1.4 Manfaat

Manfaat dari dilakukannya penelitian ini, yaitu dapat membuat *2D skeletal based animation engine* guna mempercepat proses pembuatan animasi 2D dengan konsep *skeletal-based animation*.

1.5 Batasan Masalah

Penulis membatasi masalah agar penelitian ini lebih terfokus ke dalam hal yang ingin diteliti. Adapun batasan masalah di dalam penilitian ini adalah:

1. Menggunakan API *OpenGL* versi 4.40, API *SDL* versi 2.0.8, API *KinectSDK* versi 2.0, API *ImGUI*, dan Visual Studio 2017 versi 15.7.6 dalam mengimplementasikan *2D skeletal based animation engine.*
2. Menggunakan teknologi *Kinect* v2 sebagai penerima inputan dari pengguna.
3. Penelitian ini hanya sebatas implementasi dari *animation pipeline* hingga pada tahap *global pose generation*.
4. Jumlah sendi yang digunakan berjumlah 15 yang selanjutnya akan dijelaskan pada bab perancangan
5. Fitur dari 2D *skeletal based animation engine* ini hanya sebatas transformasi pose dari pose awal menuju pose tujuan.
6. Pengujian akan dilakukan pada karakter bertipe *humanoid.*

1.6 Sistematika Pembahasan

BAB 1 PENDAHULUAN

Dalam bab ini dijelaskan latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah serta sistematika pembahasan dalam penelitian.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini menjelaskan landasan kepustakaan yang berkaitan dengan penelitian.

BAB 3 METODOLOGI PENELITIAN

Bab ini menjelaskan tahapan-tahapan dalam melakukan penelitian.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN

Bab ini menjelaskan bagaimana menganalisis kebutuhan dan perancangan arsitektur dari *animation engine* dan komponen komponen yang akan digunakan *animation engine.*

BAB 5 IMPLEMENTASI

Bab ini menjelaskan pengimplementasian komponen-komponen yang telah dirancang pada bab perancangan.

BAB 6 PENGUJIAN

Melakukan pengujian-pengujian dari komponen-komponen yang telah diimplementasi pada bab implementasi.

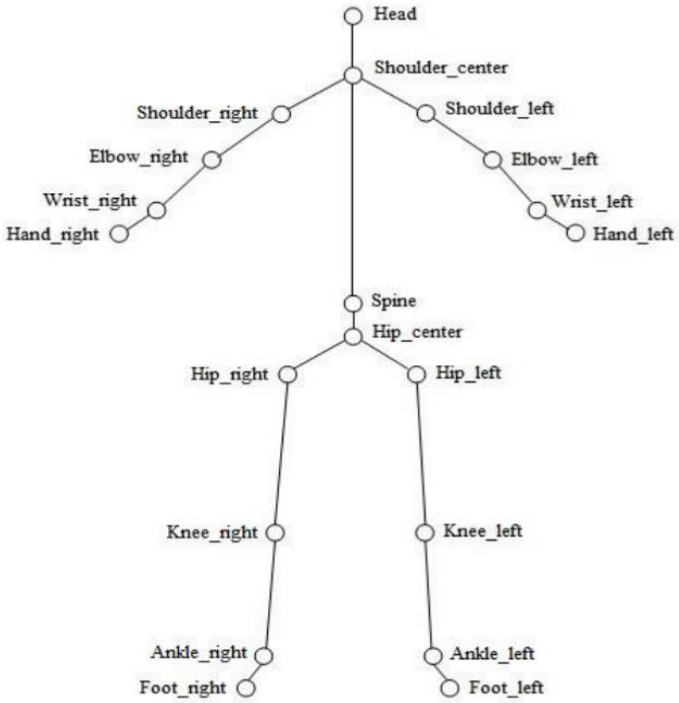
BAB 7 PENUTUP

Bab ini menjelaskan kesimpulan dari hasil pengujian yang telah dilakukan untuk menjawab rumusan masalah dan saran dari penelitian yang telah dilakukan untuk penelitian selanjutnya agar penelitian serupa dapat lebih baik.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 *Kinect*

Kinect merupakan alat penginderaan yang menyediakan pengguna fasilitas untuk berinteraksi dengan komputer dengan cara yang alami melalui *Natural User Interface* (NUI). Teknologi ini diciptakan oleh Microsoft pada tahun 2010 (Alabbasi et al., 2015). Kinect sendiri merupakan teknologi *motion capture* yang tidak memerlukan marker (*markerless*), hanya memerlukan satu kamera untuk menangkap gerakan, dan proses komputasi yang lebih cepat dibandingkan dengan teknologi *motion capture* yang biasa digunakan untuk produksi film maupun game (Shingade dan Ghotkar, 2014). Fitur fitur yang di sediakan oleh Kinect adalah kamera *red green blue* (*RGB*), *skeleton tracking, skeleton recognition, depth-image,* dan *speech recognition*. Kinect dapat mendeteksi hingga 6 tubuh dalam satu *scene*, dan tiap *skeleton* dari tubuh manusia hingga 25 *joints* dalam mode *default* (Microsoft Corporation, 2014).



Gambar 2.1 Struktur *Joint* dari *Kinect* v2

Sumber: Ashish Shingade dan Archana Ghotkar (2014)

Kinect memiliki *library* NUI sendiri yang dikeluarkan oleh Microsoft yaitu *Kinect for Windows SDK*. *Library* ini mempunyai banyak keunggulan daripada *library* NUI yang lain, mudah di pasang, tidak memerlukan kalibrasi kamera, serta dokumentasi lengkap dari SDK dan API-nya (Shingade and Ghotkar, 2014). Penggunaan Kinect tidak terbatas pada perangkat pendukung konsol game, banyak penelitian yang menggunakan teknologi ini untuk berbagai kepentingan, seperti aplikasi pembelajaran untuk memperkenalkan tari tradisional dan

melestarikannya. Pada aplikasi pembelajaran ini, pengguna dapat menirukan gerakan tari tradisional sesuai dengan pilihan, serta menggunakan sistem skor untuk menilai seberapa mirip gerakan yang diciptaakan oleh pengguna, dan mempelajari asal usul tari tradisional tersebut (Ramadijanti, Fahrul and Pangestu, 2016).

2.2 *Skeletal-Based Animation*

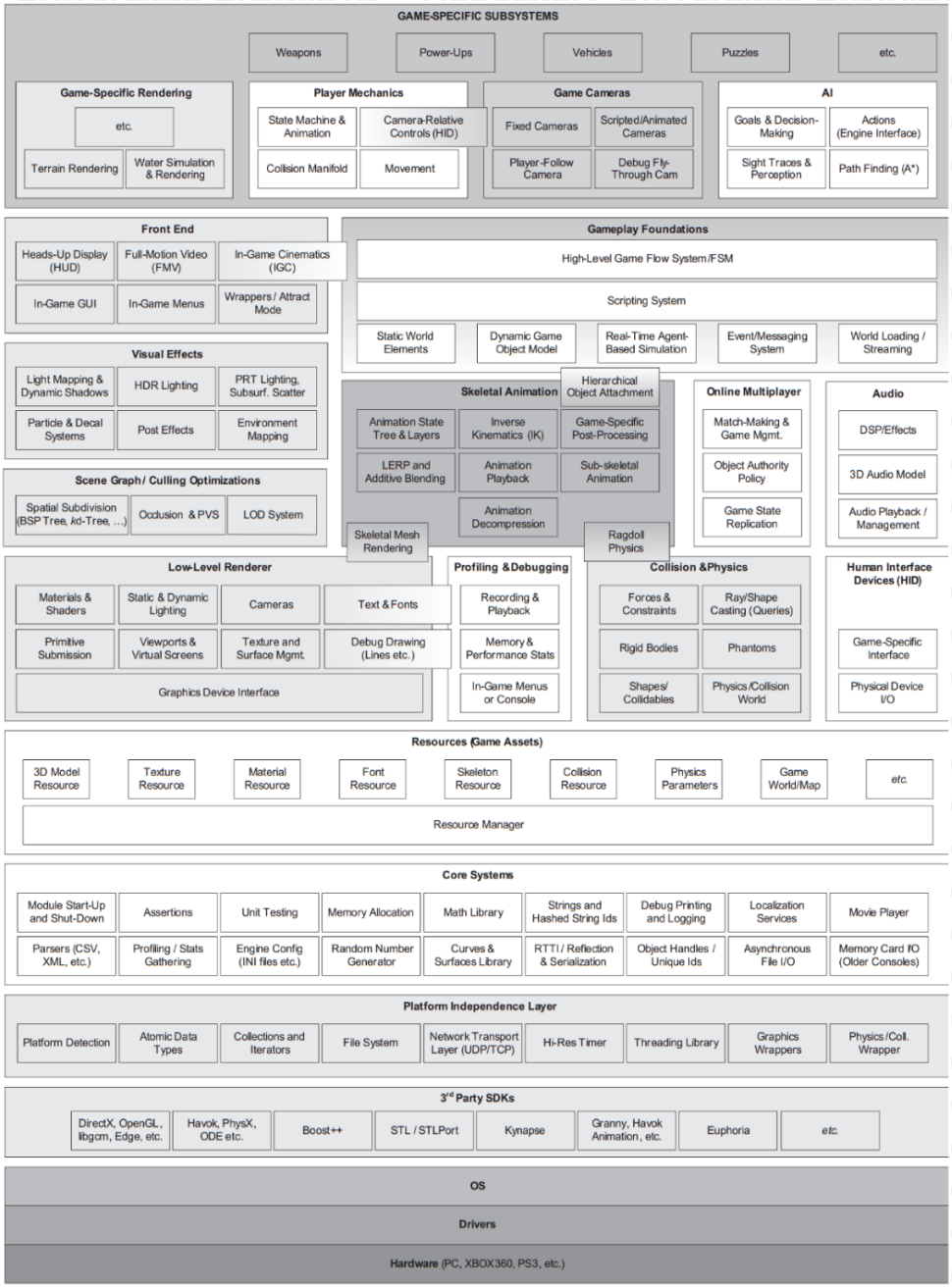
Sebuah konsep representasi ilusi virtual dari tubuh manusia yang memungkinan sebuah komputer dapat merepresentasikan gerakan alami dari manusia (Dai et al., 2010). Konsep ini sangat umum ditemukan dalam bidang animasi 2D maupun 3D guna mempermudah proses pembuatan animasi. *Skeletal Animation* sering digunakan ketika menganimasikan model 3D, pertama membuat model 3D kemudian ditambahkan dengan *skeleton* di dalam model 3D, sehingga *skeleton* terbalut dengan model 3D yang disebut dengan metode *skinning*. Setelah semua proses selesai, *skeleton* dapat diatur ke beberapa posisi yang berbeda dan bentuk dari model 3D akan mengikutinya. Untuk menggunakan konsep ini dalam 2D, gambar karakter harus di potong-potong menjadi ke beberapa bagian seperti teknik animasi *cut-out*, kemudian menambahkan *skeleton* dan mengikat gambar gambar yang berbeda ke bagian dari *skeleton* yang sesuai. Untuk menganimasikannya, cukup dengan mengubah posisi dari *skeleton* dan gambar yang terikat akan mengikuti posisi dari *skeleton* seperti konsep animasi pada 3D (Lehtonen, 2016).

Gambar 2.2 Contoh Gambar karakter 2D dengan *Skeleton*

Sumber: Jenni Lehtonen (2016) disitasi dari *Official Spine Tutorial*

2.3 Arsitektur *Animation Engine*

Dalam *animation engine*, arsitektur yang digunakan berbeda dengan *game engine* karena *animation engine* adalah bagian kecil dari *game engine*. Dapat dilihat pada Gambar 2.3, arsitektur dari *game engine* begitu kompleks yang berawal dari jenis *hardware* apa yang dituju, penggunaan driver dalam *game engine*, *software development kit* yang digunakan hingga komponen komponen yang terdapat dalam game.



Gambar 2.3 Arsitektur *Game Engine*

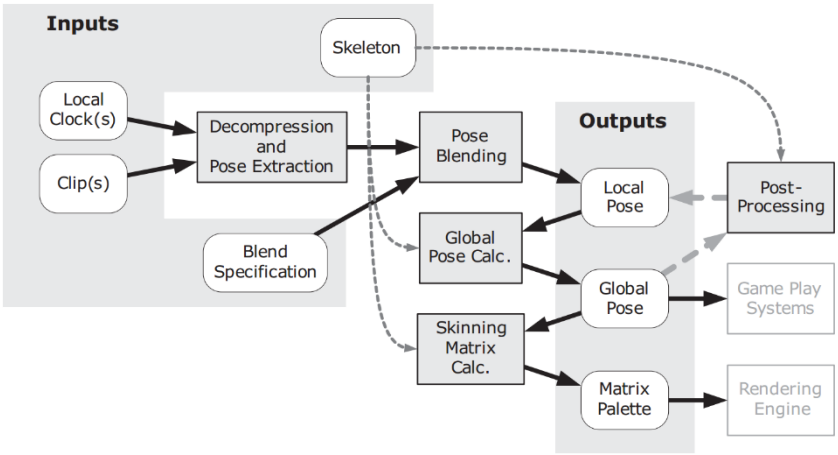
Sumber: Gregory (2018)

Arsitektur dari *animation engine* terdiri dari 3 lapisan, *animation pipeline*, *action state machine*, dan *animation controller*. Tiap lapisan mempunyai fungsi, *animation pipeline* berfungsi untuk menganimasikan setiap karakter dan objek yang ada hingga menjadi sebuah animasi. *Action state machine* berfungsi untuk memodelkan animasi yang telah dibuat menjadi sebuah *state* dengan bantuan *finite state machine* (FSM), FSM dalam animasi disebut dengan *action state machine* (ASM). *Animation controller* berfungsi sebagai *high-level system* yang mengatur animasi (Gregory, 2018).

2.3.1 *Animation Pipeline*

Menurut Gregory (2018), *animation pipeline* bertugas untuk menganimasikan setiap karakter dan objek. Arsitektur umum dari *animation pipeline* dapat dilihat pada Gambar 2.4. Pada *layer* ini, satu atau lebih klip animasi atau pose bertindak sebagai masukan beserta *blending-factor*nya, kemudian di *mix* sehingga menghasilkan animasi. Berikut adalah tahap tahap dari *animation pipeline*:

1. *Clip decompression and pose extraction*
2. *Pose blending*
3. *Global pose generation*
4. *Post-processing*
5. *Recalculation of global poses*
6. *Matrix palette generation*



Gambar 2.4 Arsitektur Umum dari *Animation Pipeline*

Sumber: Gregory (2018)

2.3.2 *Action State Machine*

Action state machine (ASM) pada *animation engine* merupakan *interface* yang bertugas untuk memetakan aksi-aksi dari karakter seperti berjalan, berlari, dan juga lompat. ASM memastikan transisi yang halus dari *state-state* dari karakter (Gregory, 2018).

2.3.3 *Animation Controller*

Animation controller adalah suatu sistem yang bertugas mengatur perilaku dari *player* maupun *non-playable character* ketika berada dalam keadaan tertentu. Contoh ketika karakter bertarung sekaligus bergerak (mode "*run-and-gun*"). *Animation controller* mengatur semua aspek dari perilaku karakter dengan animasi yang terkait (Gregory, 2018).

2.4 *Keyframe*

Frame merupakan himpunan dari data koordinat posisi dan rotasi dari masing-masing *vertex,* sedangkan *keyframe*, merupakan *frame* tertentu yang berisi data koordinat posisi dan rotasi dari masing-masing *vertex* untuk tiap sumbu x dan y. Animator lebih cenderung membuat beberapa *keyframe* untuk menghasilkan animasi daripada membuat banyak *frame* karena lebih efisien (Gregory, 2018).

2.5 *OpenGL*

OpenGL merupakan API khusus grafis untuk 2D maupun 3D. OpenGL memungkinkan *developer* untuk membuat perangkat lunak grafis dengan performa tinggi dan menarik secara visual, seperti CAD, pembuatan konten, hiburan, pengembangan game, medis, manufaktur, dan juga *virtual reality* (Khronos, 2018). OpenGL mudah untuk di pelajari, dan menyediakan berbagai fungsi yang dapat digunakan untuk mengimplementasikan grafis 2D maupun 3D, bersifat *open*, serta *multiplatform* (Seng dan Wang, 2009).

2.6 *OpenGL Mathematics*

OpenGL Mathematics (GLM) adalah sebuah library matematis untuk perangkat lunak grafis berbasis *OpenGL Shading Language (GLSL)*. *GLM* menyediakan banyak fungsi matematis dan juga kelas-kelas yang dapat diimplementasikan yang tidak disediakan oleh *GLSL*. *GLM* juga bersifat *cross-platform,* yang berati dapat digunakan pada platform manapun (OpenGL Mathematics, 2018).

2.7 *Linear Interpolation*

Menurut Gregory (2018), *linear interpolation (LERP)* adalah fungsi matematis yang umum digunakan pada *game* dan juga animasi. Pada dasarnya, fungsi *LERP* adalah seperti yang ada pada Persamaan 2.1

$$L = LERP(A, B, \beta) = (1 - \beta)A + \beta B \quad (2.1)$$

β = *blending factor*

A = komponen awal

B = komponen tujuan

Menghitung operasi *LERP* dari setiap matrix yang ada, adalah hal yang cukup rumit. Untuk menyelesaikan masalah tersebut, lebih disarankan untuk menghitung operasi *LERP* dari setiap komponen transformasi (translasi, rotasi,

skala) dari *joint*. Persamaan 2.2 adalah persamaan *LERP* untuk translasi atau dinamakan *TLERP*.

$$L = \boldsymbol{T}LERP(\boldsymbol{T}_A, \boldsymbol{T}_B, \beta) = (1 - \beta)\boldsymbol{T}_A + \beta\boldsymbol{T}_B \quad (2.2)$$

β = *Blending factor*

$\boldsymbol{T}_A$ = Koordinat awal

$\boldsymbol{T}_B$ = Koordinat tujuan

2.8 *Whitebox Testing*

Whitebox testing merupakan metode pengujian pada perangkat lunak yang berfokus pada lojik serta struktur kode dari perangkat lunak. Pengujian ini dapat dilakukan pada semua level pengembangan sistem, khususnya pengujian unit, sistem dan integrasi. Dalam pengujian ini, penguji mengetahui seluruh alur lojik dari perangkat lunak (Nidhra, 2012).

2.9 *Blackbox Testing*

Blackbox Testing merupakan suatu metode pengujian pada perangkat lunak yang berfokus pada kebutuhan fungsional dari sebuah perangkat lunak. Dalam pengujian ini, penguji hanya mengetahui masukan dan keluaran yang dibutuhkan, dengan kata lain penguji tidak mengetahui proses di dalam sistem. Contoh ideal dari metode pengujian *blackbox testing* adalah *search engine*, dimana pengguna tidak mengetahui proses yang dijalankan di dalam *search engine* (Ehmer Khan, 2011).

2.10 *Frame Rate*

Frame rate adalah tingkat kecepatan dari pemrosesan *frame* yang ditampilkan kepada penonton dan satuan yang digunakan adalah *hertz* (Hz). *Hertz* (Hz) adalah satuan yang menunjukkan jumlah siklus per detik. Satuan Hz dalam game biasanya dikenal dengan istilah *Frame Per Second* (fps) (Gregory, 2018).

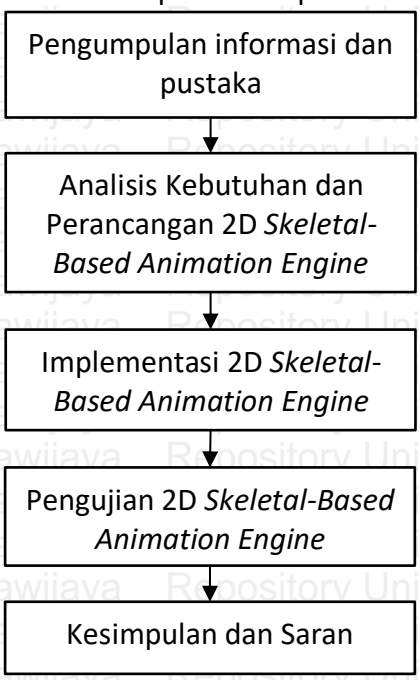
BAB 3 METODOLOGI PENELITIAN

3.1 Tipe Penelitian

Tipe penelitian dari penelitian ini adalah penelitian implementatif dengan fokus pengembangan (*Development*). Hasil akhir dari penelitian ini adalah perangkat lunak yang diharapkan dapat menjadi solusi dari masalah yang diangkat pada penilitian ini.

3.2 Strategi Penelitian

Strategi pada penilitian ini, bermula dari mengumpulkan informasi dan pustaka yang memiliki keterkaitan dengan penelitian. Informasi dan pustaka yang berkaitan dapat diperoleh melalui jurnal penelitian, buku, dan internet. Kemudian melakukan perancangan akan kelas-kelas yang nantinya digunakan untuk dapat mengimplementasikan perangkat lunak. Setelah melakukan perancangan, langkah selanjutnya adalah mengimplementasikan semua kelas yang telah dirancang. Implementasi dari penelitian ini hanya sampai pada tahap *global pose generation* yang terdapat pada lapisan *animation pipeline*. Kemudian melakukan pengujian akan fungsi utama dan kinerja dari perangkat lunak yang telah diimplementasi, dan mengambil kesimpulan serta saran berdasarkan hasil dari pengujian. Langkah-langkah dalam strategi penelitian dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Alir Metodologi Penlitian

3.3 Analisis Kebutuhan dan Perancangan *2D Skeletal-Based Animation Engine*

Pada tahap ini, dilakukan analisis kebutuhan serta perancangan komponen-komponen dari *2D Skeletal-based animation engine*. Analisis kebutuhan dilakukan

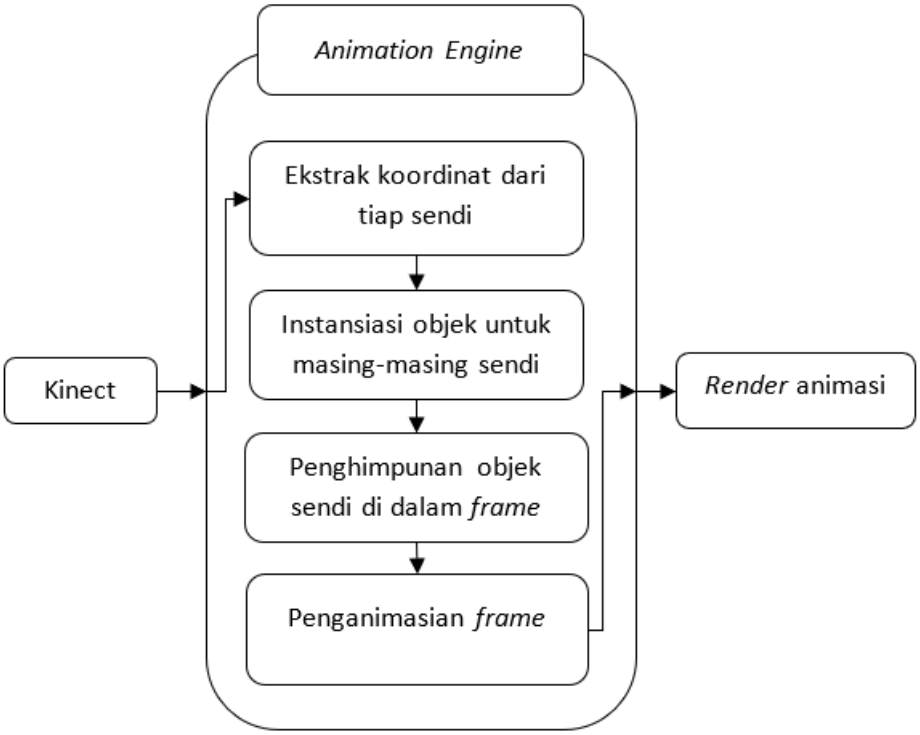
untuk menentukan fitur-fitur dari *animation engine* yang akan diimplementasikan. Perancangan dilakukan untuk merancang fitur-fitur yang telah ditentukan dari tahap analisis kebutuhan dan merancang arsitektur dari *animation engine*. Perancangan yang dilakukan adalah perancangan arsitektur dari *animation engine*, perancangan integrasi sensor *kinect* dengan *animation engine*, perancangan direktori sendi yang digunakan, dan perancangan *window workspace*.

3.3.1 Analisis Kebutuhan 2D *Skeletal-Based Animation Engine*

Analisis kebutuhan dari 2D *skeletal based animation engine* dilakukan untuk menentukan fitur-fitur dari *animation engine*. Kebutuhan-kebutuhan dari *animation engine* terdiri dari kebutuhan fungsional serta kebutuhan non-fungsional.

3.3.2 Perancangan Arsitektur dari *Animation Engine*

Perancangan arsitektur dari *animation engine* ini dimulai dengan perancangan komponen komponen inti yang akan menjadi dasar dari *animation engine*. Rancangan arsitektur *animation engine* terdapat pada Gambar 3.2



Gambar 3.2 Arsitektur *Animation Engine*

Bermula dengan bagaimana *animation engine* ini dapat membaca masukan dari pengguna. Untuk dapat membaca masukan dari pengguna berbasis NUI, diperlukan sensor Kinect yang terintegrasi dengan *animation engine*. Kemudian mengesktrak koodinat dari tiap sendi yang terdeteksi oleh sensor *Kinect*. Kemudian membuat tiap-tiap koordinat sendi menjadi objek, yang kemudian

objek tersebut di himpun dalam satu objek *frame*. *Frame* adalah representasi dari kumpulan objek-objek sendi yang telah diinstansiasi yang menyimpan koordinat dari setiap objek sendi. Kemudian menganimasikan *frame* n hingga *frame* n+1 yang telah diinstansiasi menggunakan fungsi *linear interpolation*. Fungsi utama dari *animation engine* adalah interpolasi *frame* yaitu melakukan perhitungan *linear interpolation* (LERP) antara dua *frame* agar dapat menemukan *frame* tengah di antara dua *frame*, *frame* n hingga *frame* n+1. Keluaran dari *animation engine* adalah menampilkan animasi yang sudah dihasilkan melalui perhitungan LERP. Arsitektur *animation engine* yang dirancang sesuai dengan bagian pada *animation pipeline* yaitu *clip decompression and pose extraction* yang bersesuaian dengan proses ekstrak koordinat dari tiap sendi, *pose blending* yang bersesuaian dengan proses Menganimasikan *frame*, serta *global pose generation* yang bersesuaian dengan *render* animasi.

3.3.3 Perancangan Integrasi Sensor *Kinect* dengan *Animation Engine*

Perancangan integrasi sensor *Kinect* dengan *animation engine* dilakukan agar *animation engine* dapat membaca data dari sensor *Kinect*, sehingga tidak memerlukan bantuan dari *tools* lain.

3.3.4 Perancangan Direktori Sendi

Perancangan direktori sendi dilakukan dengan tujuan untuk menyesuaikan jumlah sendi yang ada pada *Kinect* dengan yang digunakan pada *animation engine*, hal ini dilakukan karena tidak semua sendi digunakan pada *animation engine*.

3.3.5 Perancangan *Window Workspace*

Perancangan *window workspace* dilakukan untuk menyediakan ruang kerja dalam membuat animasi untuk animator. *Window workspace* memberikan tampilan visual dari gambar karakter yang telah terikat dengan *joint* yang sesuai.

3.4 Peralatan Pendukung

Untuk menghasilkan hasil perancangan, dibutuhkan perangkat keras dan perangkat lunak yang sesuai agar aplikasi dapat diimplementasikan. Pada sub bab berikut akan dijelaskan spesifikasi perangkat keras dan perangkat lunak yang digunakan untuk dapat mengimplementasikan hasil perancangan.

3.4.1 Spesifikasi Perangkat Keras

Untuk menghasilkan hasil perancangan, dibutuhkan perangkat keras guna mengimplementasikan hasil perancangan. Pada Tabel 3.1, menjelaskan perangkat keras yang digunakan untuk mengimplementasikan hasil perancangan.

Tabel 3.1 Spesifikasi Perangkat Keras

Komponen	Spesifikasi
Processor	Intel® Core™ i7-6700HQ CPU @ 2.60 GHz (8 CPUs)

Memory	8 GB
HDD	1 TB
Kinect for Windows v2.0 sensor	V2.0

3.4.2 Spesifikasi Perangkat Lunak

Spesifikasi perangkat lunak menjelaskan perangkat lunak serta *library-library* yang digunakan untuk dapat mengimplementasikan hasil perancangan. Pada Tabel 3.2, menjelaskan perangkat lunak yang digunakan untuk mengimplementasikan hasil perancangan.

Tabel 3.2 Spesifikasi Perangkat Lunak

Nama Perangkat Lunak / *Library*	Versi Perangkat Lunak / *Library*
Microsoft Visual Studio Community 2017	Versi 15.7.6
OpenGL Shader Language (GLSL)	Versi 4.40 – Build 21.20.16.4550
SDL	Versi 2.0.8
KinectSDK_2.0	Versi 2.0

3.5 Implementasi

Pada tahap ini akan dijelaskan komponen apa saja yang diimplementasikan. Untuk dapat mengimplementasikan *2D Skeletal animation engine*, diawali dengan mengimplementasikan arsitektur *animation engine*, kemudian mengintegrasikan sensor *Kinect* dengan *animation engine* serta yang terakhir adalah mengimplementasikan *window workspace*. Dalam pengimplementasian *animation engine*, implementasi menggunakan bahasa pemrograman C++, *library OpenGL* 4.4, *library SDK* 2.0.8, serta *library KinectSDK* 2.0.

3.6 Pengujian dan Analisis

Dalam tahap ini, terdapat satu skenario pengujian yaitu pengujian fungsionalitas untuk dua komponen utama dari *animation engine* yang bertujuan untuk mengetahui fungsionalitas dari *animation engine*.

3.6.1 Skenario Pengujian Fungsionalitas

Skenario pengujian fungsionalitas dilakukan untuk mengetahui fungsionalitas dari *animation engine* yaitu menganimasikan karakter dari pose awal menuju pose tujuan, serta menganimasikan karakter dari pose awal menuju pose tujuan dari file yang diimpor ke dalam *animation engine*. Pengujian dilakukan dengan menggunakan metode *blackbox testing* sehingga animasi karakter dari pose awal menuju pose tujuan dapat diketahui keberhasilannya. Skenario pengujian dapat dilihat pada Tabel 3.3

Tabel 3.3 Skenario Pengujian Fungsionalitas *Animation Engine*

No	*Parameter*	Isi
1	Nama Kasus Uji	-
	Kebutuhan Fungsional	-
	Ringkasan Kasus Uji	-
	Langkah Uji	-
	Data Masukan	-
	Hasil yang Diharapkan	-
	Hasil	-
	Status	-

Dalam melakukan pengujian, masing-masing dari komponen memiliki parameter-parameter yang harus terdeskripsikan secara padat di dalam tabel. Parameter-parameter tersebut bertujuan untuk mempermudah dalam memahami bagaimana komponen-komponen yang diuji berfungsi di dalam aplikasi yang dikembangkan.

3.7 Kesimpulan

Pengambilan kesimpulan dilakukan setelah semua pengujian dan analisis dari hasil pengujian telah selesai dilakukan. Hal ini dilakukan agar mengetahui inti dari penelitian ini.

3.8 Saran

Saran digunakan ketika semua tahap dari penelitian ini sudah selesai dilakukan dengan tujuan memperbaiki berbagai kesalahan yang terjadi selama pengujian. Dan diharapkan tidak terjadi pada pengujian dan penelitian serupa yang akan dilakukan di kemudian hari.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN

4.1 Analisis Kebutuhan dari *Animation Engine*

Untuk dapat menentukan kebutuhan-kebutuhan yang dibutuhkan oleh *animation engine*, dilakukan analisis kebutuhan untuk *animation engine*. Kebutuhan-kebutuhan yang dibutuhkan oleh *animation engine* terdiri dari kebutuhan fungsional serta kebutuhan non-fungsional.

4.1.1 Analisis Kebutuhan Fungsional

Pada kebutuhan fungsional, akan dijelaskan mengenai apa saja yang harus dapat dilakukan oleh *animation engine*. Kebutuhan-kebutuhan fungsional memiliki kode yang bersifat unik. Kode tersebut dibuat berdasarkan aturan penomoran yang memiliki struktur "AE-F-XXX", dimana AE merupakan singkatan dari *Animation Engine* yang akan dikembangkan, "F" yang merupakan representasi dari kebutuhan fungsional, serta "XXX" yang merupakan nomor urut dari kebutuhan fungsional. Hasil dari analisis kebutuhan yang didapatkan terdapat pada Tabel 4.1.

Tabel 4.1 Kebutuhan Fungsional *Animation Engine*

No	Kode Fungsi	Spesifikasi Kebutuhan
1	AE-F-001	Sistem harus dapat menganimasikan *frame* n hingga *frame* n+1.
		1.1 Sistem harus dapat menganimasikan *frame* n hingga *frame* n+1.
2	AE-F-002	Sistem harus dapat menganimasikan *frame* n hingga *frame* n+1 yang datanya berasal dari luar sistem.
		2.1 Sistem harus dapat membaca file yang diimpor ke dalam sistem.
		2.2 Sistem harus dapat menganimasikan *frame* n hingga *frame* n+1.

4.1.2 Analisis Kebutuhan Non-Fungsional

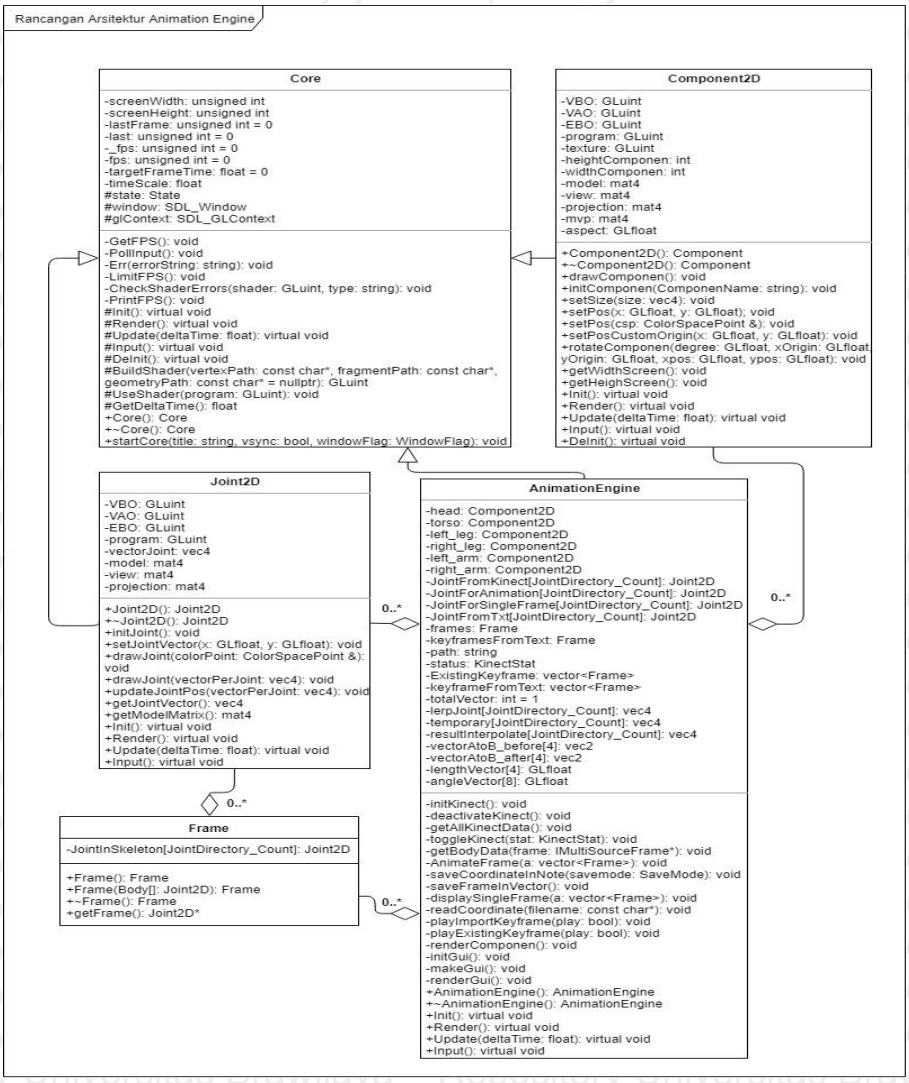
Pada kebutuhan non-fungsional, akan dijelaskan fungsi pendukung dari *animation engine* agar membantu fungsi utama dari *animation engine*. Kebutuhan-kebutuhan non-fungsional memiliki kode yang bersifat unik. Kode tersebut dibuat berdasarkan aturan penomoran yang memiliki struktur "AE-NF-000", dimana AE merupakan singkatan dari *Animation Engine* yang akan dikembangkan, "NF" yang merupakan representasi dari kebutuhan non-fungsional, serta "XXX" yang merupakan nomor urut dari kebutuhan non-fungsional. Hasil dari analisis kebutuhan yang didapatkan terdapat pada Tabel 4.2.

Tabel 4.2 Kebutuhan Non-Fungsional *Animation Engine*

No	Kode Fungsi	Definisi Kebutuhan
1	AE-NF-001	Sistem dapat mengaktifkan sensor *Kinect*
2	AE-NF-002	Sistem dapat mendapatkan informasi sendi yang ditangkap oleh sensor *Kinect*

4.2 Perancangan Arsitektur dari *Animation Engine*

Untuk dapat mengimplementasikan arsitektur dari *animation engine* dibutuhkan perancangan arsitektur dari *animation engine*, arsitektur dari *animation engine* disajikan dalam bentuk diagram kelas yang terdapat pada Gambar 4.1.



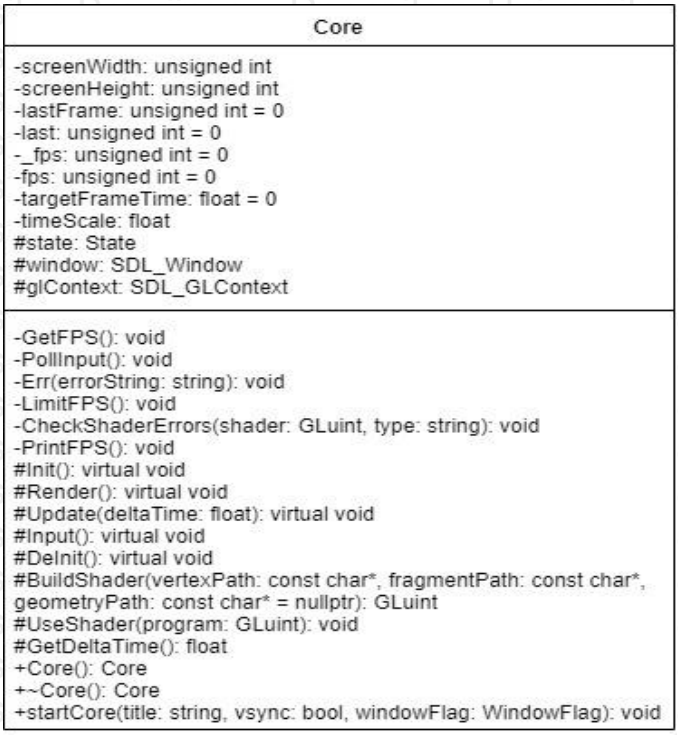
Gambar 4.1 Rancangan Kelas Diagram Arsitektur *2D Skeletal-Based Animation Engine*

Arsitektur dari *animation engine* terdiri dari beberapa kelas, yaitu kelas *Core* yang menjadi kelas dasar, serta kelas *AnimationEngine*, kelas *Joint2D*, kelas

Component2D, kelas *Bone*, dan kelas *Frame* yang menjadi kelas turunan dari kelas *Core*.

4.2.1 Perancangan Kelas *Core*

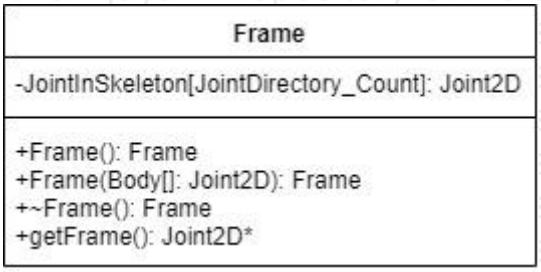
Kelas *core* merupakan kelas dasar dari *animation engine*. Pada kelas ini, dilakukan proses inisialisasi dari *SDL*, *OpenGL*, *Shader*, serta pendefinisian *engine loop*. Rancangan diagram kelas *Core* terdapat pada Gambar 4.2.



Gambar 4.2 Rancangan Diagram Kelas Core

4.2.2 Perancangan Kelas *Frame*

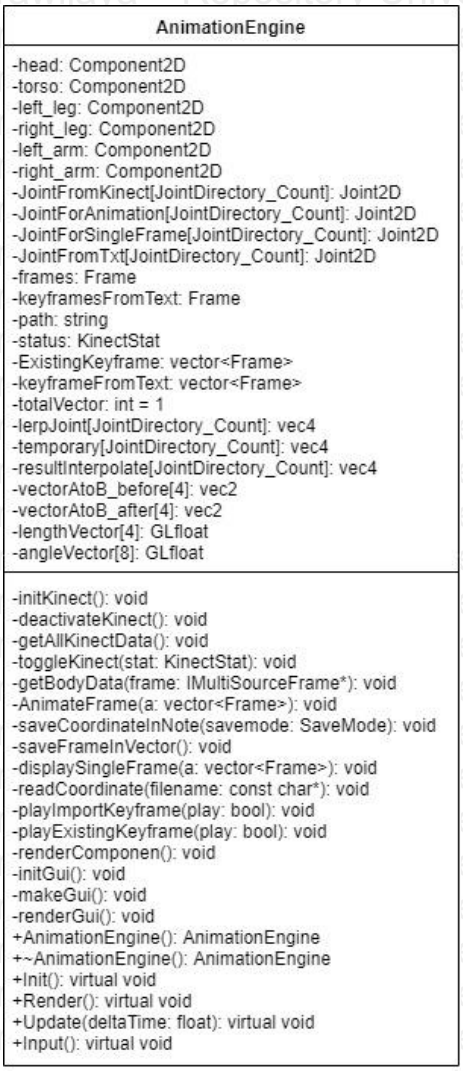
Kelas *Frame* merupakan kelas yang berfungsi menghimpun objek *Joint2D*. Memiliki konstruktor untuk dapat menginisialisasi objek *frame* dan fungsi balikan untuk mengembalikan semua objek *Joint2D* yang telah terhimpun. Rancangan diagram kelas *Frame* terdapat pada Gambar 4.3.



Gambar 4.3 Rancangan Diagram Kelas *Frame*

4.2.3 Perancangan Kelas *AnimationEngine*

Kelas *AnimationEngine* merupakan kelas turunan dari kelas *core*. Pada kelas ini fungsi integrasi sensor *Kinect*, fungsi menganimasikan *keyframe*, fungsi menyimpan *keyframe* ke dalam file .txt, serta membaca hasil *import* file .txt di definisikan. Rancangan diagram kelas *AnimationEngine* terdapat pada Gambar 4.4.

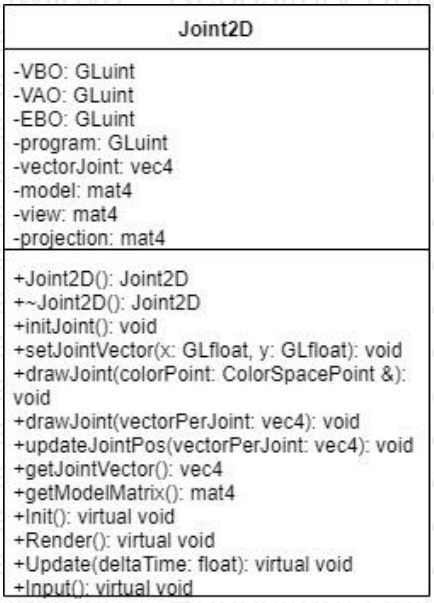


Gambar 4.4 Rancangan Diagram Kelas *AnimationEngine*

4.2.4 Perancangan Kelas *Joint2D*

Kelas *Joint2D* merupakan kelas yang merepresentasikan sendi melalui titik-titik yang ditampilkan pada *workspace*. Berawal dari inisialisasi titik, kemudian menampilkannya dalam *engine loop*. Dalam kelas ini juga dilakukan proses

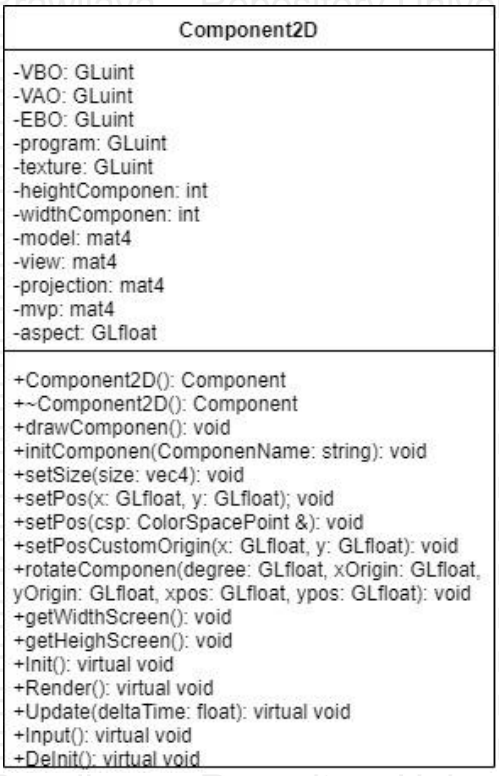
menyimpan nilai vector dari tiap sendi. Rancangan diagram kelas *Joint2D* terdapat pada Gambar 4.5.



Gambar 4.5 Rancangan Diagram Kelas *Joint2D*

4.2.5 Perancangan Kelas *Component2D*

Kelas *Component2D* merupakan kelas yang berfungsi menampilkan komponen *sprite*. Berawal dari inisialisasi komponen, kemudian menampilkannya dalam *engine loop*. Rancangan diagram kelas *Component2D* terdapat pada Gambat 4.6.



Gambar 4.6 Rancangan Diagram Kelas *Component2D*

4.3 Perancangan Integrasi Sensor *Kinect* dengan *Animation Engine*

Untuk dapat mengintegrasikan sensor *Kinect* dengan *animation engine*, diperlukan API pendukung yaitu *Kinect for Windows SDK* 2.0 yang dikeluarkan oleh *Microsoft*. Untuk dapat memahami peran dari sensor *Kinect* dengan *animation engine* dibuat gambar yang menjelaskan cara kerja dari *animation engine*. Gambar alur kerja dari *animation engine* terdapat pada Gambar 4.7.

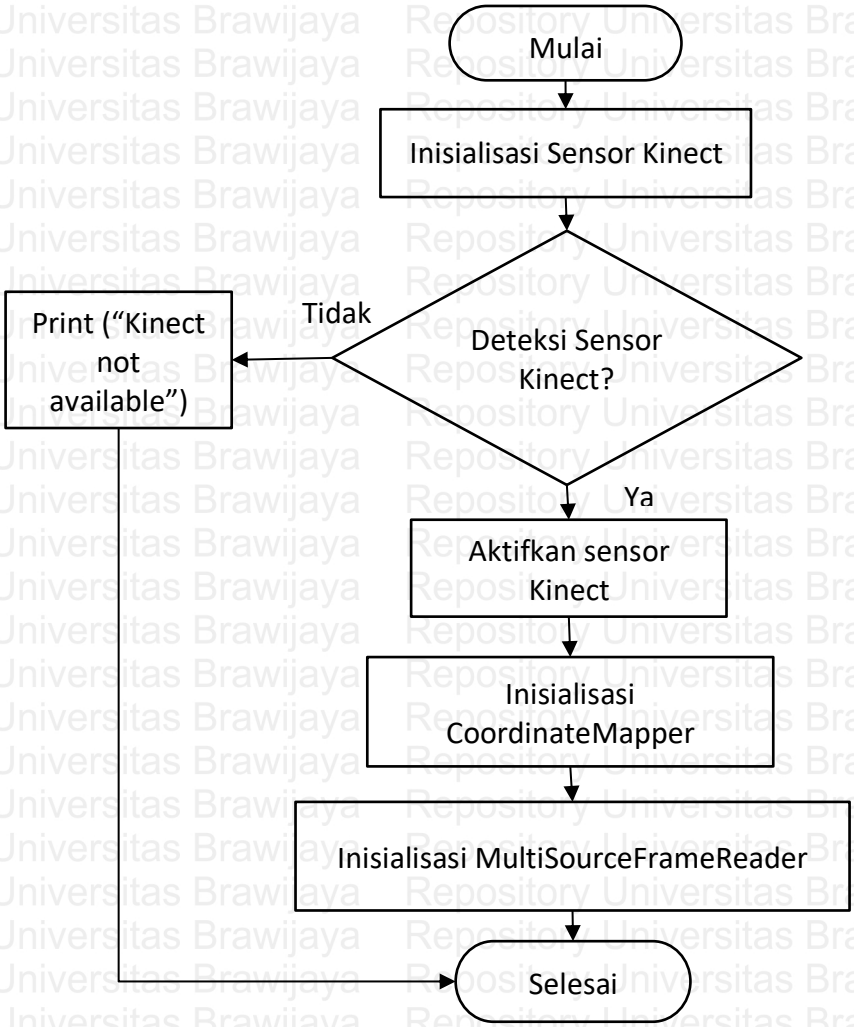
Gambar 4.7 Alur Kerja *Animation Engine* Yang Dirancang

Dari Gambar 4.7, dapat dipahami bahwa gawai Kinect menangkap gerak dari pengguna, kemudian Kinect mengolah data gerak yang didapat menjadi keluaran berupa nilai koordinat posisi sebenarnya dari sendi dan urutan sendi. Nilai koordinat posisi sendi kemudian di konversi ke dalam pixel agar dapat sesuai letaknya pada layar di dalam *animation engine*. Nilai koordinat posisi sendi yang telah di konversi dijadikan acuan letak komponen 2D. Kemudian, pengguna menyimpan nilai koordinat posisi sendi yang telah di konversi yang telah terhimpun dalam satu *frame* ke dalam vector, yang nantinya akan di lakukan proses penghitungan *linear interpolation* antar *frame* untuk menemukan *frame* antara *frame* n dan *frame* n+1. Hasil dari *linear interpolation* antar *frame* menjadi keluaran dari *animation engine* atau disebut dengan animasi.

4.3.1 Perancangan Fungsi Inisialisasi Sensor Kinect

Fungsi inisialisasi sensor Kinect adalah fungsi yang bertujuan untuk mendapatkan informasi Kinect yang terhubung dan menyalakan sensor, serta menginisialisasi *Multi Source Frame* untuk dapat menggunakan berbagai *Frame* yang disediakan oleh *Kinect*. Fungsi ini sesuai dengan kebutuhan non-fungsional

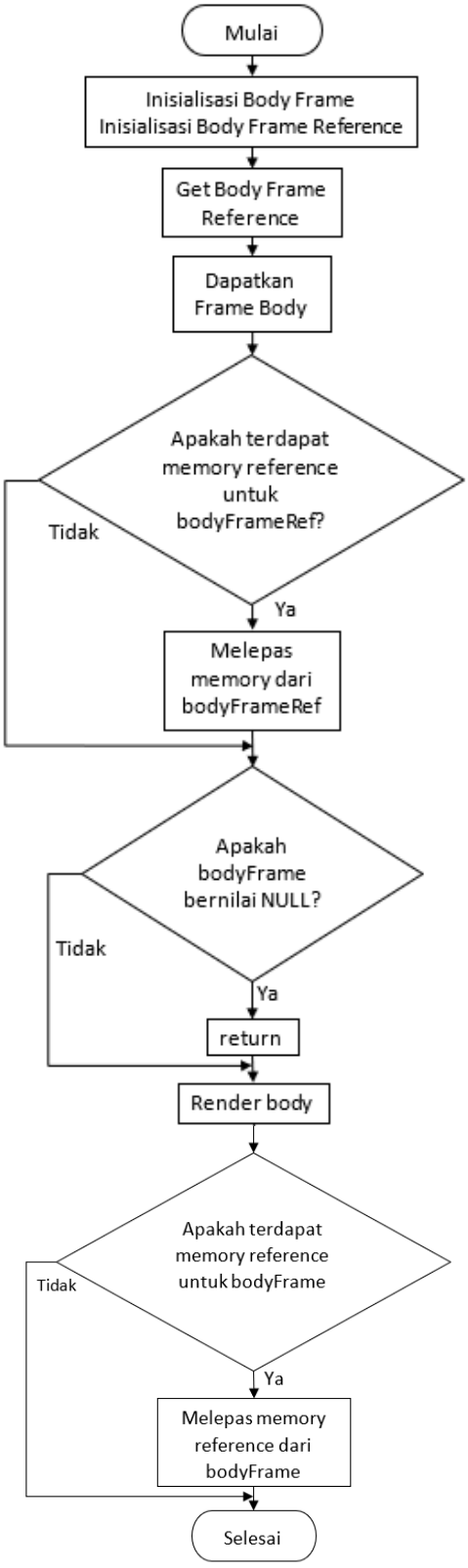
yang terdapat pada Tabel 4.2 dengan kode AE-NF-001. Proses-proses yang ada pada fungsi ini terdapat pada Gambar 4.8.



Gambar 4.8 Diagram Alir Inisialisasi Sensor *Kinect*

4.3.2 Perancangan Fungsi Deteksi Sendi Tubuh Manusia

Fungsi deteksi sendi merupakan fungsi yang berfungsi untuk mendeteksi sendi dari tubuh yang terdeteksi. Fungsi ini sesuai dengan kebutuhan non-fungsional yang terdapat pada Tabel 4.2 dengan kode AE-NF-002. Proses-proses yang ada pada fungsi ini terdapat pada Gambar 4.9.



Gambar 4.9 Diagram Alir Deteksi Sendi

Dalam fungsi deteksi sendi tubuh manusia, dilakukan inisialisasi variable pointer *Body Frame* serta *Body Frame Reference* yang bernilai null, kemudian

melakukan proses *Get Body Frame Reference*. Dari variable pointer *Body Frame Reference*, melakukan proses dapatkan *Body Frame*. Melakukan inisialisasi array *Body* sebanyak *Body_Count*. Untuk dapat mengisi array *Body*, dijalankan proses *Get and Refresh Body Data*. Kemudian menjalankan *method renderBody* dengan parameter berisi nilai memory reference dari *Body Frame*. Jika terdapat tubuh yang terdeteksi, dilakukan proses perulangan sebanyak sendi yang terdeteksi dan menampilkan sendi. Jika sendi tidak terdeteksi, maka *Body Frame* dapat dibebaskan.

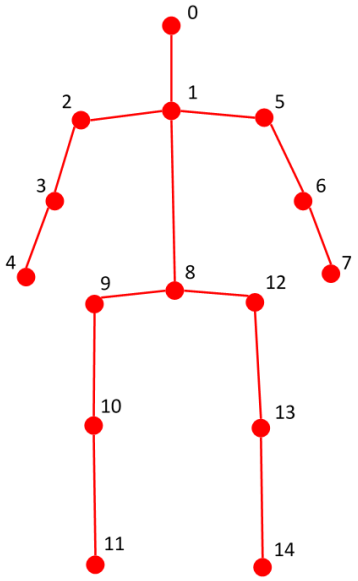
4.4 Perancangan Direktori Sendi

Sendi sendi yang digunakan, terhimpun dalam satu kelas Enumerasi dengan nama jointDirectory. Isi dari jointDirectory adalah sendi dari *Kinect* yang dipetakan sesuai dengan urutan yang ada pada Tabel 4.3. Tujuan dari pemetaan sendi adalah melakukan penyesuaian direktori sendi yang telah disediakan oleh *Kinect* karena tidak semua sendi digunakan pada *animation engine*.

Tabel 4.3 Pemetaan Sendi pada *Animation Engine*

No	Sendi pada *Animation Engine*	Sendi pada *Kinect*
0	Head	JointType_Head
1	Neck	JointType_SpineShoulder
2	ShoulderLeft	JointType_ShoulderLeft
3	ElbowLeft	JointType_ElbowLeft
4	HandLeft	JointType_HandLeft
5	ShoulderRight	JointType_ShoulderRight
6	ElbowRight	JointType_ElbowRight
7	HandRight	JointType_HandRight
8	SpineBase	JointType_SpineBase
9	HipLeft	JointType_HipLeft
10	KneeLeft	JointType_KneeLeft
11	AnkleLeft	JointType_AnkleLeft
12	HipRight	JointType_HipRight
13	KneeRight	JointType_KneeRight
14	AnkleRight	JointType_AnkleRight

Gambar 4.10 menggambarkan letak posisi dari masing masing sendi yang telah terpetakan sesuai dengan Tabel 4.3. Pada Gambar 4.10, visualisasi posisi dari

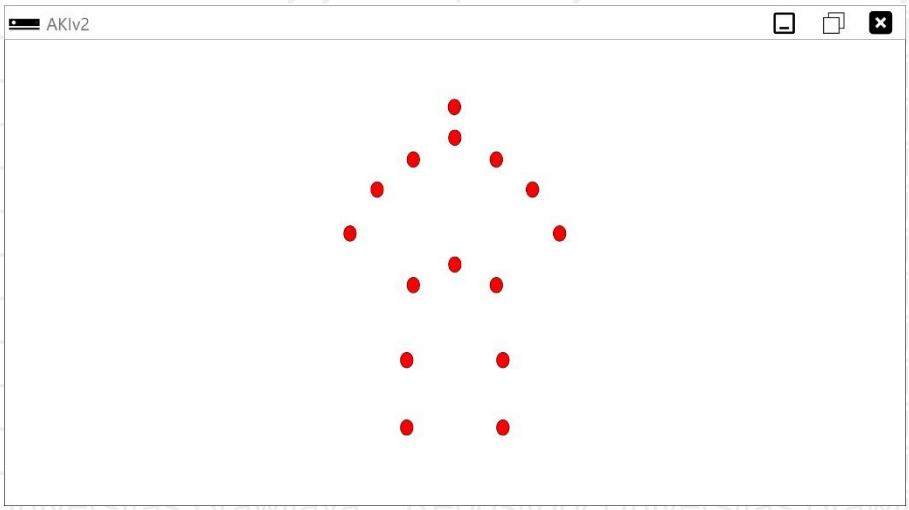


Gambar 4.10 Visualisasi Posisi dari Masing-Masing Sendi

masing-masing sendi digambarkan dengan garis yang menyambungkan satu sendi dengan sendi yang lain. Namun dalam implementasinya, garis sambung tidak di diimplementasikan dikarenakan hanya membutuhkan koordinat posisi dari masing-masing sendi.

4.5 Perancangan *Window Workspace*

Perancangan *window workspace* merupakan perancangan window utama dari *animation engine* seperti yang terdapat pada Gambar 4.11. Informasi sendi yang ditampilkan pada *window workspace* didapat dari sensor *Kinect* melalui fungsi deteksi sendi. Informasi sendi ditampilkan bertujuan untuk merepresentasikan tubuh yang terdeteksi oleh sensor *Kinect*.

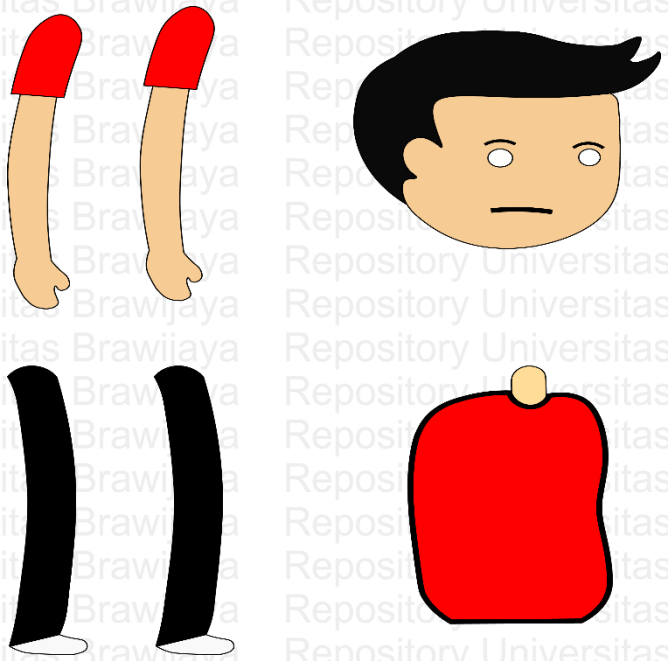


Gambar 4.11 *Mockup Animation Engine*

Di dalam *window workspace* diperlukan *user interface* agar meningkatkan interaksi pengguna dalam melakukan semua aktifitas pembuatan animasi serta gambar karakter 2D guna merepresentasikan karakter 2D dengan konsep *sketelal-based animation*. Perancangan *user interface* serta perancangan karakter 2D akan dijelaskan pada sub bab-sub bab tersendiri.

4.5.1 Perancangan Karakter 2D

Untuk dapat mengimplementasikan *animation engine* dengan konsep *skeletal-based animation*, diperlukan gambar karakter 2D. Gambar karakter 2D pada *animation engine* bertipe *humanoid*, mengingat masukan dari sensor *Kinect* merupakan gesture tubuh manusia. Untuk dapat mengimplementasikan konsep *skeletal-based animation*, bagian dari karakter akan di potong-potong sesuai dengan bagian tubuh manusia yaitu kepala, badan, lengan kanan, lengan kiri, kaki kanan, serta kaki kiri menjadi satu file tersendiri untuk masing masing bagian. Gambar karakter 2D yang telah di potong-potong sesuai dengan bagian tubuh manusia terdapat pada Gambar 4.12.



Gambar 4.12 Karakter 2D yang Telah Terpotong-potong

Sumber: https://www.kisspng.com/png-character-2d-computer-graphics-puppet-clip-art-2d-1496568/download-png.html (2018)

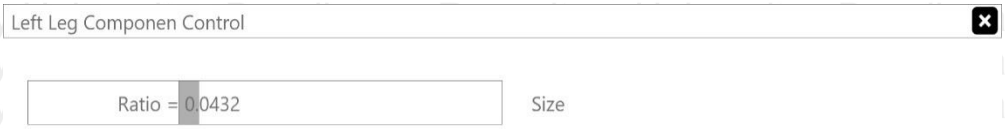
4.5.2 Perancangan *User Interface*

Untuk dapat mengatur semua aktifitas secara interaktif di dalam *window workspace*, diperlukan *user interface* dalam bentuk *graphical user interface* agar meningkatkan interaksi pengguna. Untuk *user interface* akan dirancang sebuah panel komponen dimana jumlah panel mengikuti jumlah komponen dari karakter, dan fungsi dari panel komponen adalah untuk mengatur ukuran dari komponen karakter. Diperlukan juga menu bar serta panel *main control* yang berfungsi untuk mengatur semua aktifitas dari *animation engine*. Rancangan panel komponen dari

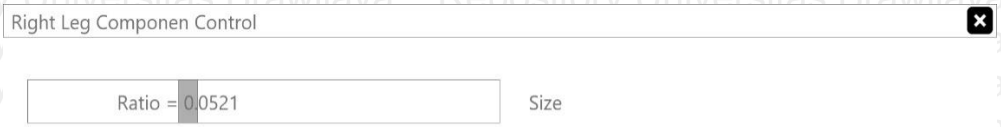
masing-masing komponen terdapat pada Gambar 4.13 untuk rancangan panel komponen *torso*, Gambar 4.14 untuk rancangan panel komponen *head*, Gambar 4.15 untuk rancangan panel komponen *left leg*, Gambar 4.16 untuk rancangan panel komponen *right leg*, Gambar 4.17 untuk rancangan panel komponen *right hand*, Gambar 4.18 untuk rancangan panel komponen *left hand*. Untuk rancangan panel *main control* terdapat pada Gambar 4.19.

Gambar 4.13 Rancangan Panel Komponen Torso

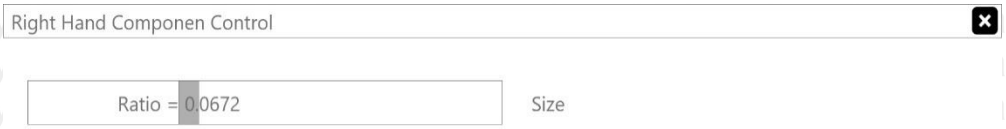
Gambar 4.14 Rancangan Panel Komponen *Head*



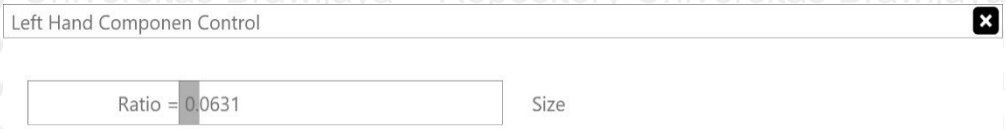
Gambar 4.15 Rancangan Panel Komponen *Left Leg*



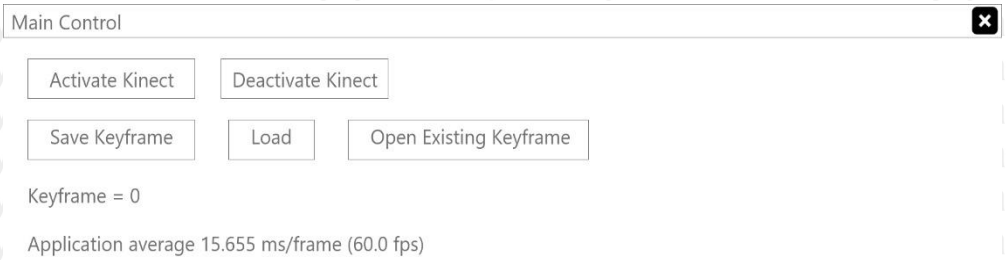
Gambar 4.16 Rancangan Panel Komponen *Right Leg*



Gambar 4.17 Rancangan Panel Komponen *Right Hand*



Gambar 4.18 Rancangan Panel Komponen *Left Hand*



Gambar 4.19 Rancangan Panel *Main Control*

4.6 Perancangan Pengujian

Perancangan pengujian merupakan perancangan yang berfokus pada bagaimana menguji komponen-komponen yang telah diimplementasikan. Pengujian akan dilakukan pada 4 komponen. 2 komponen pertama yaitu komponen inisialisasi sensor kinect dan komponen deteksi sendi tubuh manusia, komponen-komponen tersebut akan diuji dengan metode *whitebox testing* guna memastikan bahwa struktur dari komponen-komponen tersebut telah sesuai dengan perancangannya. 2 komponen selanjutnya yaitu komponen *AnimateFrame* dan *playImportKeyframe*, akan diuji dengan menggunakan metode *blackbox testing* guna memastikan apakah komponen-komponen tersebut telah diimplementasikan sesuai dengan fungsi dari komponen-komponen tersebut.

4.6.1 Perancangan Pengujian Unit Integrasi Sensor Kinect

Untuk mengetahui sensor Kinect telah terintegrasi dengan *animation engine*, diperlukan pengujian pada komponen-komponen yang telah diimplementasikan. Terdapat 2 komponen yang akan diuji dengan metode *whitebox testing* yaitu fungsi inisialisasi sensor kinect dan fungsi deteksi sendi tubuh manusia.

4.6.2 Perancangan Pengujian Fungsionalitas

Untuk mengetahui apakah *animation engine* telah berhasil diimplementasikan dengan benar, diperlukan pengujian pada komponen-komponen yang telah diimplementasikan pada *animation engine*. Terdapat 2 komponen yang akan diuji dengan menggunakan metode *blackbox testing*. Penggunaan metode *blackbox testing* digunakan untuk menilai apakah komponen-komponen yang telah diimplementasikan sesuai dengan fungsi dari komponen-komponen tersebut.

4.6.2.1 Perancangan Pengujian Fungsionalitas *Method AnimateFrame*

Tabel 4.4 Rancangan Pengujian Fungsionalitas *Method AnimateFrame*

No	*Parameter*	Isi

1	Nama Kasus Uji	Tes *method AnimateFrame*
	Kebutuhan Fungsional	Menganimasikan *frame* n hingga *frame* n+1
	Ringkasan Kasus Uji	*Animation engine* menganimasikan *frame* n hingga *frame* n+1 dengan menggunakan fungsi LERP
	Langkah Uji	- Pengguna memilih tombol *save keyframe* untuk menyimpan *keyframe* - Pengguna memilih tombol *play existing*
	Data Masukan	- *Keyframe* yang tersimpan ketika pengguna memilih tombol *save keyframe*
	Hasil yang Diharapkan	*Animation engine* menampilkan animasi dari *frame* n hingga *frame* n+1
	Hasil	-
	Status	-

Pada Tabel 4.4, terdapat rancangan pengujian komponen dari *animation engine* yaitu komponen *AnimateFrame*. Pengujian dilakukan untuk memastikan apakah komponen *AnimateFrame* mampu untuk menampilkan *frame* n hingga *frame* n+1 dengan memanfaatkan fungsi LERP, agar menghasilkan *frame* tengah antara *frame* n hingga *frame* n+1.

4.6.2.2 Perancangan Pengujian Fungsionalitas *Method playImportKeyframe*

Tabel 4.5 Rancangan Pengujian Fungsionalitas *Method playImportKeyframe*

No	*Parameter*	Isi
1	Nama Kasus Uji	Tes *method playImportKeyframe*
	Kebutuhan Fungsional	Menganimasikan *frame* n hingga *frame* n+1 dari file .txt yang diimpor ke dalam *animation engine*
	Ringkasan Kasus Uji	*Animation engine* menganimasikan *frame* n hingga *frame* n+1 dari file .txt yang diimpor ke dalam *animation engine* dengan menggunakan fungsi LERP
	Langkah Uji	- Pengguna memilih tombol *open* - Pengguna memilih tombol *from file .txt* - Pengguna memilih tombol *animate frame*
	Data Masukan	- *Keyframe* yang tersimpan di dalam file .txt yang diimpor ke dalam *animation engine*
	Hasil yang Diharapkan	*Animation engine* menampilkan animasi dari *frame* n hingga *frame* n+1 dari file .txt yang diimpor ke dalam *animation engine*
	Hasil	-
	Status	-

Pada Tabel 4.5, terdapat rancangan pengujian komponen dari *animation engine* yaitu komponen *playImportKeyframe*. Pengujian dilakukan untuk memastikan apakah fungsi *playImportKeyframe* mampu untuk menampilkan

frame n hingga *frame* n+1 dengan memanfaatkan fungsi LERP, namun *frame* yang digunakan adalah *frame* yang telah disimpan di dalam file .txt yang kemudian diimpor ke dalam *animation engine*.

BAB 5 IMPLEMENTASI

5.1 Implementasi Arsitektur *Animation Engine*

Untuk dapat mengimplementasikan arsitektur dari *animation engine*, dibuat kelas-kelas yang telah dirancang pada bab perancangan. Kelas-kelas tersebut adalah kelas *core*, kelas *Frame*, kelas *AnimationEngine*, kelas *Joint2D*, dan kelas *Component2D*. Penjelasan fungsi dari masing-masing kelas akan dijelaskan pada sub bab yang sesuai. Seluruh kode program dari arsitektur *animation engine*, terdapat pada lampiran.

5.1.1 Implementasi Kelas *Core*

Dalam kelas ini dilakukan inisialisasi fungsi fungsional dari *SDL*, pembuatan jendela aplikasi, pendefinisian *engine loop*, deklarasi fungsi untuk membangun program *shader* dan fungsi menjalankan program *shader*, serta deklarasi fungsi untuk mendapatkan nilai *fps* dari jendela aplikasi. Pada Tabel 5.1, berisi pendefinisian *engine loop* pada *animation engine*.

Tabel 5.1 Kode Program Pendefinisian *Engine Loop*

No	Core.cpp
1	//…Other Code…//
2	
3	
4	//Own Engine Loop
5	while (State::RUNNING == state)
6	{
7	float deltaTime = GetDeltaTime();
8	GetFPS();
9	/*PollInput();*/
10	Input();
11	Update(deltaTime);
12	Render();
13	SDL_GL_SwapWindow(window);
14	LimitFPS();
15	PrintFPS();
16	}
17	//…Other Code…//

Dari tabel 5.1 dapat diketahui, *engine loop* merupakan sekumpulan fungsi yang dijalankan dalam perulangan, baris 5 merupakan deklarasi perulangan dengan kondisi dimana *state* bernilai *running*. Di dalam perulangan, terdapat fungsi *GetFPS* yang berfungsi untuk mendapatkan nilai *fps*, fungsi *Input* yang menangani semua masukan dari pengguna, fungsi *update* yang berfungsi menangani segala bentuk perubahan dari komponen-komponen yang ada di dalam engine, fungsi *Render* yang berfungsi untuk menyajikan tampilan visual dari komponen, *SDL_GL_SwapWindow*(*window*) yang befungsi untuk menukar buffer, fungsi *LimitFPS* yang berfungsi untuk membatasi nilai *fps* yang di dapatkan, kemudian *PrintFPS* yang berfungsi untuk menampilkan informasi *fps* kepada pengguna. Seluruh kode program dari kelas *Core*, terdapat pada lampiran.

5.1.2 Implementasi Kelas *Frame*

Kelas ini merupakan kelas realisasi dari konsep *Frame* dan berfungsi untuk menghimpun objek dari *Joint2D* dan menyimpan koordinat dari masing-masing objek *Joint2D* yang terhimpun. Implementasi dari rancangan kelas *Frame* terdapat pada Tabel 5.2 dalam bentuk kode program.

Tabel 5.2 Kode Program *Header* dari Kelas *Frame*

No	Frame.h
1	#pragma once
2	#include "Joint2D.h"
3	class Frame
4	{
5	public:
6	Frame();
7	Frame(Joint2D body[]);
8	~Frame();
9	
10	Joint2D * getFrame();
11	private:
12	Joint2D JointInSkeleton[JointDirectory_Count];
13	};

Baris 2 merupakan proses memasukkan *header* dari kelas *Joint2D*, hal ini dilakukan agar kelas *Joint2D* dapat digunakan di dalam kelas *Frame*. Baris 6 merupakan konstruktor standar dari kelas *Frame*. Baris 7 merupakan overloading konstruktor dari kelas *Frame* dengan parameter array objek dari *Joint2D* ber-*access modifier* publik. Overloading konstruktor dideklarasikan dengan maksud agar objek dari kelas *Frame* dapat secara langsung menyimpan objek dari array objek *Joint2D*. Kemudian pada baris 10, terdapat fungsi dengan *access modifier* publik, *getFrame()* yang berfungsi mengembalikan objek dari *Joint2D* yang sudah disimpan pada saat menginstansiasi objek *Frame*. Pada baris 12, terdapat deklarasi variable bertipe data array *Joint2D JointInSkeleton* berjumlah *JointDirectory_Count*, ber-*access modifier* privat. Seluruh kode program dari kelas *Frame*, terdapat pada lampiran.

5.1.3 Implementasi Kelas *AnimationEngine*

Kelas ini merupakan kelas realisasi dari *Core* yang memiliki fungsi inisialisasi dari *Kinect* agar dapat mengaktifkan sensor *Kinect* dan fungsi deteksi badan manusia. Implementasi dari rancangan kelas *AnimationEngine* terdapat pada Tabel 5.3 dalam bentuk kode program.

Tabel 5.3 Kode Program *Header* dari Kelas *AnimationEngine*

No	AnimationEngine.h
1	#pragma once
2	#include "Core.h"
3	#include "Component2D.h"
4	#include "Joint2D.h"
5	#include "Frame.h"
6	#include "Bone.h"
7	
8	enum class KinectStat { ACTIVE, DEACTIVE };
9	enum class SaveMode { OVERWRITE, SAVE_EXISTING };
10	
11	class AnimationEngine :

```
12          public Core
13  {
14  public:
15          AnimationEngine();
16          ~AnimationEngine();
17          virtual void Init();
18          virtual void Render();
19          virtual void Update(float deltaTime);
20          virtual void Input();
21
22  private:
23          void initKinect();
            //Initialize Kinect
24          void deactivateKinect();
            //Deactivate Kinect
25          void getAllKinectData();
            //Get All Data Stream From Kinect
26          void toggleKinect(KinectStat stat);
            //Toggle Kinect
27          void getBodyData(IMultiSourceFrame* frame);          //Get
    Body Data From Stream
28          void AnimateFrame(vector<Frame> a);
            //Animating Keyframe
29          void saveCoordinateInNote(SaveMode savemode);  //Save
    Coordinate To .TXT File
30          void saveFrameInVector();
            //Save Frame in Vector //Like Array List
31          void displaySingleFrame(vector<Frame> a);
            //Display Single Frame
32          void readCoordinate(const char * fileName);          //Read
    Coordinate From .TXT File
33          void playImportKeyframe(bool play);
            //Play Imported Keyframe
34          void playExistingKeyframe(bool play);                //Play
    Directly
35          void renderComponen();
            //Render Componen
36          void initGui();
                  //Initialize GUI
37          void makeGui();
                  //Make GUI
38          void renderGui();
            //Render GUI
39
40          //User define object
41          Component2D head, torso, left_leg, right_leg, left_arm,
    right_arm;
42          Joint2D JointFromKinect[JointDirectory_Count];
43          Joint2D JointForAnimation[JointDirectory_Count];
44          Joint2D JointForSingleFrame[JointDirectory_Count];
45          Joint2D JointFromTxt[JointDirectory_Count];
46          Bone ConnectingBone[JointDirectory_Count - 1];
47          Frame frames, keyframesFromText;
48          string path;
49          KinectStat status;
50
51          //Make Vector Frame
52          vector<Frame> ExistingKeyframe;
53          vector<Frame> keyframeFromText;
54
55          int totalVector = 1;
56
57          vec4 lerpJoint[JointDirectory_Count];
58          vec4 temporary[JointDirectory_Count];
59          vec4 resultInterpolate[JointDirectory_Count];
```

```
60
61          vec2 vectorAtoB_before[4];
62          vec2 vectorAtoB_after[4];
63
64          GLfloat lengthVector[4];
65
66          GLfloat angleVector[8];
67  };
```

Fungsi inisialisasi *Kinect* terdapat pada baris 23, dan fungsi deteksi badan manusia terdapat pada baris 27. Seluruh kode program dari kelas *AnimationEngine* terdapat pada lampiran.

5.1.4 Implementasi Kelas *Joint2D*

Kelas ini adalah kelas yang merepresentasikan sendi di dalam *animation engine*, memiliki fungsi untuk inisialisasi sendi dan menyimpan nilai vektor dari masing masing sendi yang terdeteksi. Implementasi dari rancangan kelas *Joint2D* terdapat pada Tabel 5.4 dalam bentuk kode program.

Tabel 5.4 Kode Program *Header* dari Kelas *Joint2D*

No	Joint2D.h
1	`#pragma once`
2	`#include "Core.h"`
3	`class Joint2D :`
4	`        public Core`
5	`{`
6	`public:`
7	`        Joint2D();`
8	`        ~Joint2D();`
9	`        void initJoint();`
10	`        void setJointVector(GLfloat x, GLfloat y);`
11	`        void drawJoint(ColorSpacePoint & colorPoint);`
12	`        void drawJoint(vec4 vectorPerJoint);`
13	`        void updateJointPos(vec4 vectorPerJoint);`
14	`        vec4 getJointVector();`
15	`        mat4 getModelMatrix();`
16	
17	`        virtual void Init();`
18	`        virtual void Render();`
19	`        virtual void Update(float deltaTime);`
20	`        virtual void Input();`
21	
22	`private:`
23	`        GLuint VBO, VAO, EBO, program;`
24	`        vec4 vectorJoint;`
25	`        mat4 model, view, projection;`
26	`};`

Fungsi inisialisasi sendi terdapat pada baris 9 dengan akses *modifier* publik, dan fungsi untuk menyimpan nilai vektor dari sendi terdapat pada baris 10 dengan akses *modifier* publik. Baris 11 dan 12 merupakan fungsi untuk menggambar sendi. Namun terdapat perbedaan, yaitu pada parameter yang digunakan. Pada baris 23, terdapat deklarasi variabel VBO, VAO, EBO, serta program dengan tipe data GLuint. Pada baris 24, terdapat deklarasi variabel vectorJoint dengan tipe data vec4. Pada baris 25, terdapat deklarasi variabel model, view, serta projection dengan tipe data mat4. Seluruh kode program dari kelas *Joint2D* terdapat pada lampiran.

5.1.5 Implementasi Kelas *Component2D*

Kelas ini bertugas untuk menampilkan *sprite* dari karakter yang akan ditampilkan. Memiliki fungsi inisialisasi *sprite* dan menampilkannya pada jendela aplikasi. Implementasi dari rancangan kelas *Component2D* terdapat pada Tabel 5.5 dalam bentuk kode program.

Tabel 5.5 Kode Program *Header* dari Kelas *Component2D*

No	Component2D.h
1	#pragma once
2	#include "Core.h"
3	class Component2D :
4	public Core
5	{
6	public:
7	Component2D();
8	~Component2D();
9	void drawComponen();
10	void initComponen(string ComponenName);
11	void setSize(vec4 size);
12	void setPos(GLfloat x, GLfloat y);
13	void setPos(ColorSpacePoint & csp);
14	void setPosCustomOrigin(GLfloat x, GLfloat y);
15	void rotateComponen(GLfloat degree, GLfloat xOrigin, GLfloat yOrigin, GLfloat xpos, GLfloat ypos);
16	int getWidthScreen();
17	int getHeightScreen();
18	virtual void Init();
19	virtual void Render();
20	virtual void Update(float deltaTime);
21	virtual void Input();
22	virtual void DeInit();
23	
24	private:
25	GLuint VBO, VAO, EBO, texture, program;
26	int heightComponen, widthComponen;
27	mat4 model, view, projection, mvp;
28	GLfloat aspect;
29	};

Fungsi inisialisasi *sprite* terdapat pada baris 10 berparameter nama komponen bertipe data string, dan fungsi menampilkannya terdapat pada baris 9. Kedua fungsi tersebut berakses *modifier* publik. Pada baris 25, terdapat deklarasi variabel VBO, VAO, EBO, texture, serta program dengan tipe data GLuint. Pada baris 26, terdapat deklarasi variabel heightComponen dan widthComponen dengan tipe data integer. Pada baris 27, terdapat deklarasi variabel model, view, projection, dan mvp dengan tipe data matriks 4x4. Pada baris 28, terdapat deklarasi variabel aspect dengan tipe data GLfloat. Seluruh kode program dari kelas *Component2D* terdapat pada lampiran.

5.2 Integrasi sensor *Kinect* dengan *Animation Engine*

Untuk dapat mengintegrasikan sensor Kinect dengan *animation engine* agar dapat menggunakan seluruh fitur yang disediakan oleh sensor *Kinect*, diperlukan header dari API *Kinect*, *Kinect.h* pada file *Core.h*. Untuk pengaktifan perangkat *Kinect*, dilakukan inisialisasi fungsi pengaktifan perangkat *Kinect* pada kelas *AnimationEngine*. Implementasi integrasi sensor Kinect dengan *animation engine*

terdapat pada Tabel 5.6 dalam bentuk kode program pada baris 4, dan agar dapat menggunakan API *Kinect*, diperlukan juga menyertakan header dari *Windows.h* dan *Ole2.h*.

Tabel 5.6 Kode Program Integrasi Sensor Kinect dengan *Animation Engine*

No	Core.h
1	#pragma once
2	#include <Windows.h>
3	#include <Ole2.h>
4	#include <Kinect.h>
5	#include <gl3w/GL/gl3w.h>
6	#include <SOIL/SOIL.h>
7	#include <string>
8	#include <fstream>
9	#include <sstream>
10	#include <iostream>
11	#include <iterator>
12	#include <nlohmann/json.hpp>
13	#include <vector>
14	#define GLM_ENABLE_EXPERIMENTAL
15	#include <glm/glm/glm.hpp>
16	#include <glm/glm/gtc/matrix_transform.hpp>
17	#include <glm/glm/gtc/type_ptr.hpp>
18	#include <glm/glm/gtx/vector_angle.hpp>
19	#include <SDL/SDL.h>
20	#include <imgui/imgui.h>
21	#include <imgui/imgui_impl_opengl3.h>
22	#include <imgui/imgui_impl_sdl.h>

5.2.1 Implementasi Fungsi Inisialisasi Sensor *Kinect*

Untuk dapat mengimplementasikan fungsi inisialisasi sensor *Kinect,* dilakukan deklarasi objek dari sensor beserta objek-objek pendukung dengan lingkup *global*. Deklarasi fungsi dilakukan pada file *AnimationEngine.h*, dan untuk pendefinisiannya dilakukan pada file *AnimationEngine.cpp*. Pendefinisian fungsi inisialisasi sensor Kinect terdapat pada Tabel 5.7 dalam bentuk kode program.

Tabel 5.7 Kode Program Fungsi Inisialisai Sensor Kinect

No	void AnimationEngine::initKinect()
1	...Other Code...
2	//Activate Kinect
3	void AnimationEngine::initKinect()
4	{
5	GetDefaultKinectSensor(&sensor);
6	if (sensor != NULL)
7	{
8	sensor->Open();
9	sensor->get_CoordinateMapper(&mapper);
10	sensor->OpenMultiSourceFrameReader(
11	FrameSourceTypes_Color |
12	FrameSourceTypes_Body,
13	&multiSourceReader
14	);
15	}
16	else
17	{
18	printf("Kinect Not Available!\n");
19	}
20	}
21	...Other Code...

Pada bagian awal kode, dilakukan proses untuk mendapatkan nilai dari sensor, kemudian mengecek nilai dari sensor, jika sensor diaktifkan dengan menjalankan fungsi *Open*(). Karena *animation engine* dapat mendeteksi tubuh manusia, maka sensor perlu menjalankan fungsi untuk mendapatkan *get_CoordinateMapper()* dengan parameter *mapper*. Sensor harus membuka pembaca *frame* bertipe `FrameSourceTypes_Color` untuk mendapatkan informasi *frame* warna dari kamera pada sensor *Kinect*, dan pembaca *frame* bertipe `FrameSourceTypes_Body` untuk mendapatkan informasi tubuh. Jika sensor bernilai `NULL`, maka pada *console* akan tercetak "*Kinect Not Available!*".

5.2.2 Implementasi Fungsi Deteksi Sendi Tubuh Manusia

Untuk mengimplementasikan fungsi deteksi sendi tubuh manusia, dilakukan pendeklarasian fungsi *getBodyData* dengan parameter `IMultiSourceFrame` pada file *AnimationEngine.h*. Untuk pendefinisian fungsi, dilakukan pada file *AnimationEngine.cpp*. Pendefinisian fungsi deteksi sendi tubuh manusia terdapat pada Tabel 5.8 dalam bentuk kode program.

Tabel 5.8 Kode Program Fungsi Deteksi Sendi Tubuh Manusia

No	void AnimationEngine::getBodyData(IMultiSourceFrame * frame)
1	//Get Skeleton Data from Kinect
2	void AnimationEngine::getBodyData(IMultiSourceFrame * frame)
3	{
4	IBodyFrame* bodyFrame;
5	IBodyFrameReference* bodyFrameRef = NULL;
6	frame->get_BodyFrameReference(&bodyFrameRef);
7	bodyFrameRef->AcquireFrame(&bodyFrame);
8	
9	if (bodyFrameRef) bodyFrameRef->Release();
10	if (!bodyFrame) return;
11	renderBody(bodyFrame);
12	if (bodyFrame) bodyFrame->Release();
13	}

Pada awal pendefinisian fungsi, dilakukan deklarasi variabel `IBodyFrame* bodyFrame` serta `IBodyFrameReference* bodyFrameRef`, selanjutnya melakukan proses mendapatkan nilai `bodyFrame` dengan menggunakan fungsi `AcquireFrame()`. Kemudian melakukan proses pengecekan apakah sensor mendeteksi tubuh. Jika sensor *Kinect* mendeteksi tubuh, maka *animation engine* akan menampilkan sendi sesuai dengan letak yang didapatkan dari sensor *Kinect*. Jika tubuh tidak terdeteksi, maka `bodyFrame` melepas memori agar memori yang teralokasikan dapat digunakan oleh proses yang lain.

5.3 Implementasi Direktori Sendi

Dalam mengimplementasikan direktori sendi yang sesuai, dibuat sebuah kelas Enumerasi yang berisi urutan indeks dari sendi-sendi yang telah disesuaikan.

Tabel 5.9 Deklarasi Enumerasi *jointDirectory*

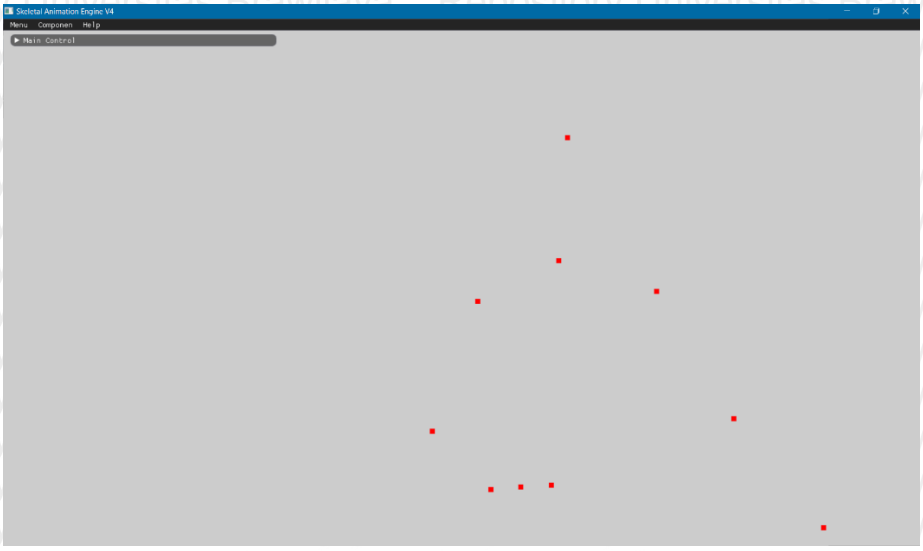
No	enum jointDirectory
1	enum jointDirectory
2	{
3	Head = 0,
4	Neck = 1,

```
5           ShoulderLeft = 2,
6           ElbowLeft = 3,
7           HandLeft = 4,
8           ShoulderRight = 5,
9           ElbowRight = 6,
10          HandRight = 7,
11          SpineBase = 8,
12          HipLeft = 9,
13          KneeLeft = 10,
14          AnkleLeft = 11,
15          HipRight = 12,
16          KneeRight = 13,
17          AnkleRight = 14,
18          JointDirectory_Count = (AnkleRight + 1)
19  };
```

Pada Tabel 5.9, dalam deklarasi enum *jointDirectory* pemberian index sendi dimulai dari dari 0 hingga 14. Dimulai dari *head* bernilai 0, *neck* bernilai 1, *ShoulderLeft* bernilai 2, *ElbowLeft* bernilai 3, *HandLeft* bernilai 4, *ShoulderRight* bernilai 5, *ElbowRight* bernilai 6, *HandRight* bernilai 7, *SpineBase* bernilai 8, *HipLeft* bernilai 9, *KneeLeft* bernilai 10, *AnkleLeft* bernilai 11, *HipRight* bernilai 12, *KneeRight* bernilai 13, *AnkleRight* bernilai 14, serta *JointDirectroy_Count* yang bernilai 15. *JointDirectory_Count* adalah anggota dari enum *jointDirectory* yang menyimpan nilai total dari anggota enum.

5.4 Implementasi *Window Workspace*

Window workspace merupakan jendela utama dari aplikasi *animation engine* ini dan untuk mengimplementasikannya, dilakukan instansiasi objek window dari *library SDL*. Hasil tampilan dari implementasi *window workspace* terdapat pada Gambar 5.1.



Gambar 5.1 Hasil Implementasi *Window Workspace*

5.4.1 Implementasi *User Interface*

Dalam mengimplementasikan *user interface*, digunakan API untuk *user interface* yang khusus diperuntukkan untuk aplikasi yang dikembangkan

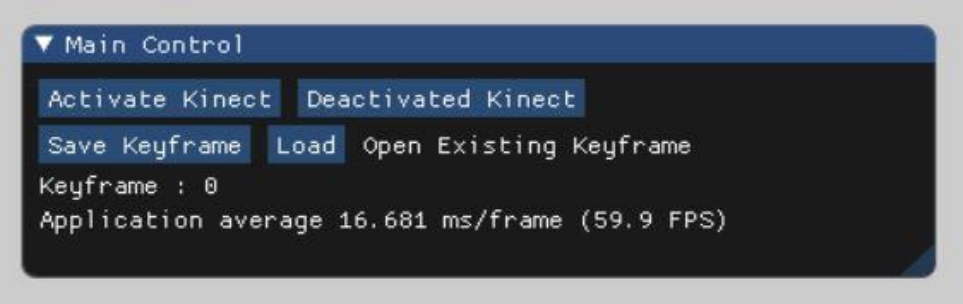
menggunakan *OpenGL*, yaitu *ImGui*. Kode program implementasi *user interface* terdapat pada Tabel 5.10.

Tabel 5.10 Kode Program Implementasi *User Interface*

No	AnimationEngine.cpp
584	//...Other Code...//
585	
586	//Initialize GUI
587	void AnimationEngine::initGui(){...}
606	
607	//Make GUI Here <<---- ADD HERE
608	void AnimationEngine::makeGui(){...}
816	
817	//Render GUI
818	void AnimationEngine::renderGui()

Implementasi *user interface* dilakukan di dalam kelas *AnimationEngine* di dalam fungsi initGui() yang berfungsi untuk menginisialisasi API ImGui pada baris 587, kemudian fungsi makeGui() yang berfungsi untuk mendeklarasikan *user interface* yang akan ditampilkan di dalam *window workspace* pada baris 608. Pada fungsi makeGui(), dideklarasikan panel komponen untuk komponen dari karakter dan panel *main control*. Kemudian fungsi renderGui() yang berfungsi untuk menampilkan *user interface* yang telah dideklarasikan pada fungsi makeGui() pada baris 818. Hasil dari kode program implementasi *user interface* untuk panel komponen, terdapat pada Gambar 5.2. Hasil implementasi dari kode program untuk *user interface* panel *main control*, terdapat pada Gambar 5.3.

Gambar 5.2 Hasil Implementasi Kode Program *User Interface* untuk Panel Komponen



Gambar 5.3 Hasil Implementasi Kode Program *User Interface* untuk *Main Control*

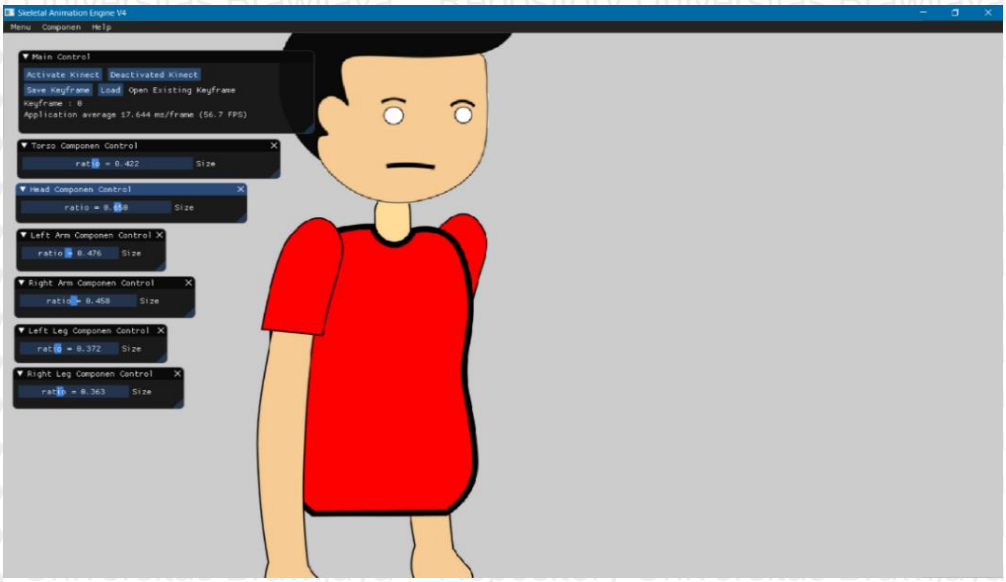
5.4.2 Implementasi Karakter 2D

Untuk mengimplementasikan karakter 2D, pada *AnimationEngine.cpp* dilakukan instansiasi objek untuk masing-masing bagian karakter, kepala, tubuh, tangan_kanan, tangan_kiri, kaki_kiri, kaki_kanan. Kode program implementasi karakter 2D terdapat pada Tabel 5.11.

Tabel 5.11 Kode Program Implementasi Karakter 2D

No	AnimationEngine.cpp
77	//Initialize 2D Component
78	{
79	kepala.initComponen("head.png");
80	tubuh.initComponen("torso.png");
81	tangan_kanan.initComponen("left_arm.png");
82	tangan_kiri.initComponen("left_arm.png");
83	kaki_kanan.initComponen("left_leg.png");
84	kaki_kiri.initComponen("left_leg.png");
85	}

Hasil akhir dari implementasi *window workspace* di mana *user interface* serta karakter 2D telah diimplementasikan, terdapat pada Gambar 5.4.



Gambar 5.4 Hasil Akhir Implementasi *Window Workspace*

BAB 6 PENGUJIAN

6.1 Pengujian Unit Integrasi Sensor Kinect

Pada sub bab pengujian unit, dilakukan pengujian terhadap dua *method* yang telah dirancang pada tahap perancangan dan diimplementasi pada tahap implementasi. Dua *method* yang akan diuji adalah *method initKinect* serta *method getBodyData*. Dalam pengujian ini digunakan metode pengujian *whitebox testing*.

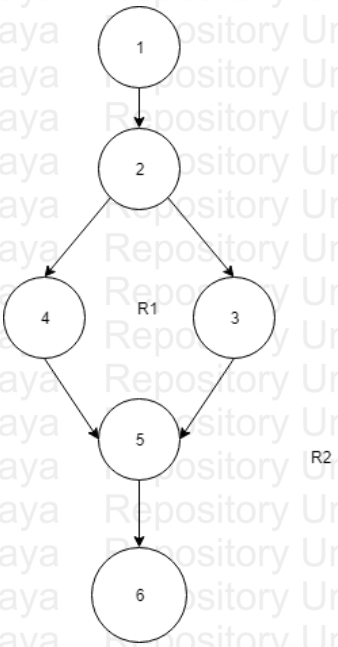
6.1.1 Pengujian Unit *Method initKinect*

Pseudocode dari *method initKinect* dituliskan pada Tabel 6.1 dibawah ini. Baris dari tiap kode diberi nomor sesuai dengan pembagian *node*.

Tabel 6.1 *Pseudocode Method initKinect*

No	Pseudocode Method initKinect	
1	Call method GetDefaultKinectSensor	1
2	If sensor is not equal to NULL	2
3	Call method sensor Open	3
4	Call method sensor get_CoordinateMapper	
5	Call method sensor OpenMultiSourceFrameReader	
6	Else	4
7	Call method printf with "Kinect Not Available!\n"	
8	EndIf	5
9	End	6

Berdasarkan *pseudocode* dari *method initKinect* pada Tabel 6.1, dibuat sebuah *flow graph* yang terdapat pada Gambar 6.1.



Gambar 6.1 *Flow Graph Method initKinect*

Berdasarkan Gambar 6.1, dapat diketahui jumlah *independent path* dengan menggunakan perhitungan *cyclomatic complexity*. Perhitungan *cyclomatic complexity* atau V(G) dari *method initKinect* adalah:

- V(G) = total *edge* – total *node* + 2

  V(G) = 6 – 6 + 2 = 2
- V(G) = total *predicate node* + 1

  V(G) = 1 + 1 = 2
- V(G) = total *region*

  V(G) = 2

Dari hasil perhitungan *cyclomatic complexity* yang telah dilakukan, dapat disimpulkan bahwa *method initKinect* memiliki 2 *independent path*. *Path-path* tersebut antara lain:

- *Path* 1 = 1 – 2 – 3 – 5 – 6
- *Path* 2 = 1 – 2 – 4 – 5 – 6

Berdasarkan penetuan jalur independent, maka dapat dibuat 2 kasus uji dalam pengujian unit *method initKinect*. Hasil dari pengujian unit untuk *method initKinect* terdapat pada Tabel 6.2.

Tabel 6.2 Hasil Pengujian Unit *Method initKinect*

Path	**Kasus Uji**	**Hasil**		**Status**
		Ekspektasi	**Aktual**	
1	*Method initKinect* dijalankan dengan kondisi sensor Kinect telah terinisialisasi dan terdeteksi oleh *animation engine*	Sensor kinect terdeteksi oleh *animation engine* yang kemudian mengaktifkan sensor Kinect, menginisialisasi MultiSourceFrameReader, serta inisialisasi CoordinateMapper	Sensor kinect terdeteksi oleh *animation engine* yang kemudian mengaktifkan sensor Kinect, menginisialisasi MultiSourceFrameReader, serta inisialisasi CoordinateMapper	Pass
2	*Method initKinect* dijalankan dengan kondisi sensor Kinect telah terinisialisasi namun	Sensor kinect tidak terdeteksi oleh *animation engine*, dan mencetak "*Kinect not available*" pada *console*	Sensor kinect tidak terdeteksi oleh *animation engine*, dan mencetak "*Kinect not available*" pada *console*	Pass

	sensor kinect tidak terdeteksi oleh *animation engine*			

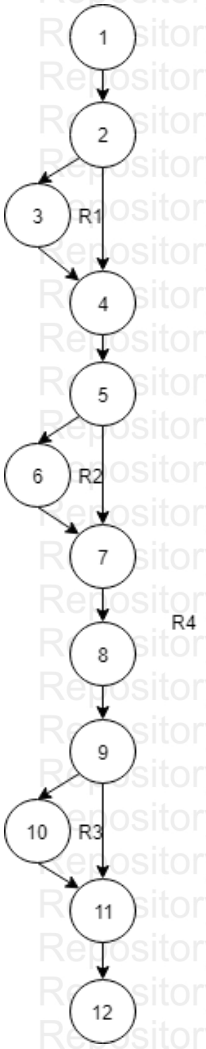
6.1.2 Pengujian Unit *Method getBodyData*

Pseudocode dari *method getBodyData* dituliskan pada Tabel 6.3 dibawah ini. Baris dari tiap kode diberi nomor sesuai dengan pembagian *node*.

Tabel 6.3 *Pseudocode Method getBodyData*

No	Pseudocode Method getBodyData	
1	IBodyFrame* bodyFrame;	1
2	Set IBodyFrameReference bodyFrameRef to NULL	
3	Call method frame get_BodyFrameReference	
4	Call method bodyFrameRef AcquireFrame	2
5	If bodyFrameRef	3
6	Call method bodyFrameRef Release	4
7	EndIf	5
8	If not bodyFrame	6
9	Return	7
10	EndIf	8
11	Call method renderBody with bodyFrame	9
12	If bodyFrame	10
13	Call method bodyFrame Release	11
14	EndIf	12
15	End	

Berdasarkan *pseudocode* dari *method getBodyData* pada Tabel 6.3, dibuat sebuah *flow graph* yang terdapat pada Gambar 6.2.



Gambar 6.2 *Flow Graph Method getBodyData*

Berdasarkan Gambar 6.2, dapat diketahui jumlah *independent path* dengan menggunakan perhitungan *cyclomatic complexity*. Perhitungan *cyclomatic complexity* atau V(G) dari *method getBodyData* adalah:

- V(G) = total *edge* - total *node* + 2
  V(G) = 14 – 12 + 2 = 4
- V(G) = total *predicate node* + 1
  V(G) = 3 + 1 = 4
- V(G) = total *region*
  V(G) = 4

Dari hasil perhitungan *cyclomatic complexity* yang telah dilakukan, dapat disimpulkan bahwa *method getBodyData* memiliki 4 *independent path*. *Path-path* tersebut antara lain:

- *Path* 1 = 1 – 2 – 4 – 5 – 7 – 8 – 9 – 11 – 12

- *Path* 2 = 1 – 2 – 3 – 4 – 5 – 7 – 8 – 9 – 11 – 12
- *Path* 3 = 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 11 – 12
- *Path* 4 = 1 – 2 – 3 – 4 – 5 – 7 – 8 – 9 – 10 – 11 – 12

Berdasarkan penetuan jalur independent, maka dapat dibuat 2 kasus uji dalam pengujian unit *method getBodyData*. Hasil dari pengujian unit untuk *method getBodyData* terdapat pada Tabel 6.4.

Tabel 6.4 Hasil Pengujian Unit *Method getBodyData*

Path	**Kasus Uji**	**Hasil**		**Status**
		Ekspektasi	**Aktual**	
1	*Method getBodyData* dijalankan, namun *animation engine* tidak me-*render* sendi manusia	*Animation engine* tidak merender sendi, memori dari variable *bodyFrameRef* tidak di bebaskan serta memori dari variable *bodyFrame* tidak dilepaskan	*Animation engine* tidak merender sendi, memori dari variable *bodyFrameRef* tidak di bebaskan serta memori dari variable *bodyFrame* tidak dilepaskan	Pass
2	*Method getBodyData* dijalankan, memori dari *bodyFrameRef* di bebaskan namun *animation engine* tidak me-*render* sendi manusia	*Animation engine* tidak me-*render* sendi, memori dari variable *bodyFrameRef* di bebaskan namun memori dari variable *bodyFrame* tidak dilepaskan	*Animation engine* tidak me-*render* sendi, memori dari variable *bodyFrameRef* di bebaskan namun memori dari variable *bodyFrame* tidak dilepaskan	Pass
3	*Method getBodyData* dijalankan, memori dari *bodyFrameRef* di bebaskan, memori dari bernilai null, *animation engine* tidak me-*render* sendi manusia	*Animation engine* tidak me-*render* sendi karena variabel *bodyFrame* bernilai null yang menyebabkan *method getBodyData* dihentikan, namun memori dari variable *bodyFrameRef* di bebaskan dan memori dari variable *bodyFrame* tidak dilepaskan	*Animation engine* tidak me-*render* sendi karena variabel *bodyFrame* bernilai null yang menyebabkan *method getBodyData* dihentikan, namun memori dari variable *bodyFrameRef* di bebaskan dan memori dari variable *bodyFrame* tidak dilepaskan	Pass

4	*Method getBodyData* dijalankan, memori dari *bodyFrameRef* di bebaskan, memori dari *bodyFrame* tidak bernilai null, *animation engine* me-*render* sendi manusia	*Animation engine* me-*render* sendi karena variabel dari *bodyFrame* tidak bernilai null. Memori dari variabel *bodyFrame* serta *bodyFrameRef* dilepaskan	*Animation engine* me-*render* sendi karena variabel dari *bodyFrame* tidak bernilai null. Memori dari variabel *bodyFrame* serta *bodyFrameRef* dilepaskan	Pass

6.2 Pengujian Fungsionalitas pada *Animation Engine*

Fungsionalitas pada *animation engine* diuji dengan menggunakan metode *blackbox testing*. Pengujian ini berfungsi untuk menguji validasi dari fungsi *AnimateFrame*, dan fungsi *playImportKeyframe*.

6.2.1 Pengujian Fungsionalitas *Method AnimateFrame*

Fungsi *AnimateFrame* diuji dengan menggunakan metode *blacbox testing.* Hasil dari pengujian untuk *method AnimateFrame* terdapat pada Tabel 6.5.

Tabel 6.5 Hasil Pengujian Fungsional *Method AnimateFrame*

No	*Parameter*	Isi
1	Nama Kasus Uji	Tes *method AnimateFrame*
	Kebutuhan Fungsional	Menganimasikan *frame* n hingga *frame* n+1
	Ringkasan Kasus Uji	*Animation engine* menganimasikan *frame* n hingga *frame* n+1 dengan menggunakan fungsi LERP
	Langkah Uji	- Pengguna memilih tombol *save keyframe* untuk menyimpan *keyframe* - Pengguna memilih tombol *play existing*
	Data Masukan	- *Keyframe* yang tersimpan ketika pengguna memilih tombol *save keyframe*
	Hasil yang Diharapkan	*Animation engine* menampilkan animasi dari *frame* n hingga *frame* n+1
	Hasil	*Animation engine* menampilkan animasi dari *frame* n hingga *frame* n+1
	Status	*VALID*

Untuk menguji *method AnimateFrame* kasus uji yang digunakan yaitu menganimasikan *frame* n hingga *frame* n+1 dengan memanfaatkan fungsi LERP,

dan hasil dari pengujian fungsi tersebut dapat menampilkan animasi dari *frame* n hingga *frame* n+1. Hasil tersebut sesuai dengan hasil yang diharapkan, dan status valid dari *method AnimateFrame* adalah *VALID.*

6.2.2 Pengujian Fungsionalitas *Method playImportKeyframe*

Fungsi *AnimateFrame* diuji dengan menggunakan metode *blacbox testing*. Hasil dari pengujian untuk *method AnimateFrame* terdapat pada Tabel 6.6.

Tabel 6.6 Hasil Pengujian Fungsionalitas *Method playImportKeyframe*

No	*Parameter*	Isi
1	Nama Kasus Uji	Tes *method playImportKeyframe*
	Kebutuhan Fungsional	Menganimasikan *frame* n hingga *frame* n+1 dari file .txt yang diimpor ke dalam *animation engine*
	Ringkasan Kasus Uji	*Animation engine* menganimasikan *frame* n hingga *frame* n+1 dari file .txt yang diimpor ke dalam *animation engine* dengan menggunakan fungsi LERP
	Langkah Uji	- Pengguna memilih tombol *open* - Pengguna memilih tombol *from file .txt* - Pengguna memilih tombol *animate frame*
	Data Masukan	- *Keyframe* yang tersimpan di dalam file .txt yang diimpor ke dalam *animation engine*
	Hasil yang Diharapkan	*Animation engine* menampilkan animasi dari *frame* n hingga *frame* n+1 dari file .txt yang diimpor ke dalam *animation engine*
	Hasil	*Animation engine* menampilkan animasi dari *frame* n hingga *frame* n+1 dari file .txt yang diimpor ke dalam *animation engine*
	Status	*VALID*

Untuk menguji fungsi *playImportKeyframe* kasus uji yang disediakan yaitu menganimasikan *frame* n hingga *frame* n+1 dari file berekstensi .txt yang diimpor ke dalam *animation engine* dengan memanfaatkan fungsi LERP, dan hasil dari pengujian fungsi tersebut adalah dapat menampilkan animasi dari *frame* n hingga *frame* n+1 dari file berekstensi .txt yang diimpor ke dalam *animation engine*. Hasil tersebut sesuai dengan hasil yang diharapkan, dan nilai status dari *method playImportKeyframe* adalah *VALID*. Dari pengujian yang telah dilakukan pada *method AnimateFrame* serta *method playImportKeyframe*, didapatkan hasil valid pada kedua *method* tersebut, hasil tersebut menunjukkan bahwa fungsionalitas dari *animation engine* telah diimplementasi serta berfungsi sesuai dengan kebutuhan dari *animation engine*.

BAB 7 PENUTUP

7.1 Kesimpulan

Dari penelitian ini dapat ditarik kesimpulan yaitu:

1. Arsitektur dari 2D *skeletal based animation engine* terdiri dari beberapa proses, yang berawal dari proses ektraksi koordinat dari tiap sendi yang di dapat melalui sensor Kinect, kemudian menginstansiasi objek untuk masing-masing sendi. Lalu masuk pada proses penghimpunan objek sendi yang telah diinstansiasi menjadi satu objek *frame*. Kemudian melakukan proses animasi *frame* yang selanjutnya ditampilkan pada layar. Untuk dapat merealisasikan proses-proses tersebut dibutuhkan beberapa kelas, yaitu kelas Core yang bertindak sebagai kelas dasar. Kemudian kelas AnimationEngine yang merupakan kelas realisasi dari kelas Core dan memiliki fungsionalitas untuk mendapatkan masukan dari sensor *Kinect*, memproses setiap keyframe yang kemudian dijalankan dalam bentuk animasi. Kelas Joint2D merupakan kelas yang berfungsi sebagai representasi dari sendi yang di dapat dari sensor Kinect, dan kelas Component2D yang berfungsi untuk menampilkan komponen gambar 2D, sehingga terlihat bagaimana bentuk implementasi dari teknik animasi *skeletal-based*.
2. Dari hasil pengujian yang telah dilakukan, didapatkan hasil pengujian dengan nilai valid. Nilai valid yang didapatkan menunjukkan bahwa *animation engine* telah berhasil diimplementasikan dan sesuai dengan kebutuhan.

7.2 Saran

Dari penelitian yang telah dilakukan, terdapat beberapa saran yang perlu dipertimbangkan agar *animation engine* ini dapat dikembangkan lebih lanjut yaitu:

1. Penambahan window pemilihan karakter untuk karakter bertipe *non-humanoid*.
2. Menambahkan fungsi kliping agar dapat mengkombinasikan dua animasi atau lebih.
3. Menambahkan editor untuk *sprite* agar dapat memodifikasi *sprite*.

DAFTAR PUSTAKA

Alabbasi, H., Gradinaru, A., Moldoveanu, F. dan Moldoveanu, A., 2015. Human motion tracking & evaluation using Kinect V2 sensor. In: *2015 E-Health and Bioengineering Conference (EHB)*. [online] 2015 E-Health and Bioengineering Conference (EHB). Iasi, Romania: IEEE.pp.1–4. Tersedia melalui: IEEExplore < https://ieeexplore.ieee.org/document/7391465> [Diakses 5 Oktober 2018]

Dai, H., Cai, B., Song, J. dan Zhang, D., 2010. Skeletal Animation Based on BVH Motion Data. In: *2010 2nd International Conference on Information Engineering and Computer Science*. [online] 2010 2nd International Conference on Information Engineering and Computer Science. Wuhan, China: IEEE.p.4. Tersedia melalui: IEEExplore < https://ieeexplore.ieee.org/document/5678292> [Diakses 5 Oktober 2018]

Ehmer Khan, Mohd., 2011. Different Approaches To Black box Testing Technique For Finding Errors. *International Journal of Software Engineering & Applications*, [online] Tersedia di: < https://www.researchgate.net/publication/268419508_Different_Approaches_To_Black_box_Testing_Technique_For_Finding_Errors> [Diakses 9 Maret 2019]

Gregory, J., 2018. *Game Engine Architecture*. 3rd ed. Florida: A K Peters/CRC Press

Khronos, 2018. Khronos. [online] Tersedia di: < https://www.khronos.org/opengl/> [Diakses 20 Desember 2018]

Lehtonen, J., 2016. FROM 2D-SPRITE TO SKELETAL ANIMATIONS. *FROM 2D-SPRITE TO SKELETAL ANIMATIONS – Boosting the performance of a mobile application*, [online] Tersedia di: < https://www.theseus.fi/bitstream/handle/10024/113773/Lehtonen_Jenni.pdf > [Diakses 17 November 2018]

Microsoft Corporation, 2014. Human Interface Guidelines v2.0. [pdf] Microsoft Corporation. Tersedia di: < http://download.microsoft.com/download/6/7/6/676611B4-1982-47A4-A42E-4CF84E1095A8/KinectHIG.2.0.pdf> [Diakses 18 November 2018]

Nidhra, S., 2012. Black Box and White Box Testing Techniques - A Literature Review. *International Journal of Embedded Systems and Applications*, [online] Tersedia di: < https://www.researchgate.net/publication/276198111_Black_Box_and_White_Box_Testing_Techniques_-_A_Literature_Review> [Diakses 9 Maret 2019]

OpenGL Mathematics, 2018. OpenGL Mathematic. [online] Tersedia di: < https://glm.g-truc.net/0.9.9/index.html> [Diakses 20 Desember 2018]

Pan, J. dan Zhang, J.J., 2011. Sketch-Based Skeleton-Driven 2D Animation and Motion Capture. In: Z. Pan, A.D. Cheok and W. Müller, eds. *Transactions on Edutainment VI*. [online] Berlin, Heidelberg: Springer Berlin

Heidelberg.pp.164–181. Tersedia melalui: Springer < https://link.springer.com/chapter/10.1007%2F978-3-642-22639-7_17> [Diakses 15 Oktober 2018]

Ramadijanti, N., Fahrul, H.F. dan Pangestu, D.M., 2016. Basic dance pose applications using kinect technology. In: *2016 International Conference on Knowledge Creation and Intelligent Computing (KCIC)*. [online] 2016 International Conference on Knowledge Creation and Intelligent Computing (KCIC). Manado, Indonesia: IEEE.pp.194–200. Tersedia melalui: IEEExplore < https://ieeexplore.ieee.org/document/7883646> [Diakses 5 Oktober 2018]

Seng, D. dan Wang, H., 2009. Realistic Real-Time Rendering of 3D Terrain Scenes Based on OpenGL. In: *2009 First International Conference on Information Science and Engineering*. [online] 2009 First International Conference on Information Science and Engineering. Nanjing, China: IEEE.pp.2121–2124. Tersedia melalui: IEEExplore < https://ieeexplore.ieee.org/document/5454581> [Diakses 18 November 2018]

Shingade, A. dan Ghotkar, A., 2014. Animation of 3D Human Model Using Markerless Motion Capture Applied To Sports. *International Journal of Computer Graphics & Animation*, [online] Tersedia di: < https://www.researchgate.net/publication/260147325_Animation_of_3D_Human_Model_Using_Markerless_Motion_Capture_Applied_To_Sports> [Diakses 18 November 2018]

LAMPIRAN A KODE PROGRAM

A.1 *Class Core*

A.1.1 Core.h

```
No  Core.h
1   #pragma once
2   #include <Windows.h>
3   #include <Ole2.h>
4   #include <Kinect.h>
5   #include <gl3w/GL/gl3w.h>
6   #include <SOIL/SOIL.h>
7   #include <string>
8   #include <fstream>
9   #include <sstream>
10  #include <iostream>
11  #include <iterator>
12  #include <nlohmann/json.hpp>
13  #include <vector>
14  #define GLM_ENABLE_EXPERIMENTAL
15  #include <glm/glm/glm.hpp>
16  #include <glm/glm/gtc/matrix_transform.hpp>
17  #include <glm/glm/gtc/type_ptr.hpp>
18  #include <glm/glm/gtx/vector_angle.hpp>
19  #include <SDL/SDL.h>
20  #include <imgui/imgui.h>
21  #include <imgui/imgui_impl_opengl3.h>
22  #include <imgui/imgui_impl_sdl.h>
23  //#include <glfw/include/GLFW/glfw3.h>
24  //#include <glew/include/GL/glew.h>
25
26  #pragma region CoreClass
27  using namespace std;
28  using namespace glm;
29  using namespace nlohmann;
30  using json = nlohmann::json;
31
32  enum class State { RUNNING, EXIT };
33  enum class WindowFlag { WINDOWED, FULLSCREEN,
    EXCLUSIVE_FULLSCREEN, BORDERLESS };
34
35  const int width = 1920;
36  const int height = 1080;
37
38  class Core
39  {
40  public:
41          Core();
42          ~Core();
43          void startCore(string title, bool vsync, WindowFlag
    windowFlag);
44
45  protected:
46          virtual void Init() = 0;
47          virtual void Render() = 0;
48          virtual void Update(float deltaTime) = 0;
49          virtual void Input() = 0;
50          virtual void DeInit();
51          GLuint BuildShader(const char* vertexPath, const char*
    fragmentPath, const char* geometryPath = nullptr);
52          void UseShader(GLuint program);
53          State state;
```

```
54          float GetDeltaTime();
55          SDL_Window* window;
56          SDL_GLContext glContext;
57
58  private:
59          unsigned int screenWidth, screenHeight, lastFrame = 0, last
    = 0, _fps = 0, fps = 0;
60          float targetFrameTime = 0, timeScale;
61          void GetFPS();
62          void PollInput();
63          void Err(string errorString);
64          void LimitFPS();
65          void CheckShaderErrors(GLuint shader, string type);
66          void PrintFPS();
67  };
68  #pragma endregion
```

A.1.2 Core.cpp

```
No      Core.cpp
1       #include "Core.h"
2
3       Core::Core()
4       {
5       }
6
7       Core::~Core()
8       {
9       }
10
11      //Start Core
12      void Core::startCore(string title, bool vsync, WindowFlag
        windowFlag)
13      {
14              SDL_GL_SetAttribute(SDL_GL_CONTEXT_FLAGS, 0);
15              SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,
        SDL_GL_CONTEXT_PROFILE_CORE);
16              SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 4);
17              SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 4);
18
19              /*glewExperimental = GL_TRUE;*/
20              //Init SDL
21              SDL_Init(SDL_INIT_EVERYTHING);
22
23              //Tell SDL Doublebuffer
24              SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
25
26              Uint32 flags = SDL_WINDOW_OPENGL;
27
28              if (WindowFlag::EXCLUSIVE_FULLSCREEN == windowFlag) {
29                      flags |= SDL_WINDOW_FULLSCREEN;
30              }
31
32              if (WindowFlag::FULLSCREEN == windowFlag) {
33                      flags |= SDL_WINDOW_FULLSCREEN_DESKTOP;
34              }
35
36              if (WindowFlag::BORDERLESS == windowFlag) {
37                      flags |= SDL_WINDOW_BORDERLESS;
38              }
39
40              if (WindowFlag::WINDOWED == windowFlag)
41              {
42                      flags |= SDL_WINDOW_RESIZABLE;
43              }
```

```
44
45          //Setup SDL Window
46          window = SDL_CreateWindow(title.c_str(),
        SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, width, height,
        flags);
47          if (window == nullptr)
48          {
49              Err("Failed to create SDL window!");
50          }
51
52          glContext = SDL_GL_CreateContext(window);
53          if (glContext == nullptr)
54          {
55              Err("Failed to create SDL_GL context!");
56          }
57
58          //Set up gl3w
59          GLenum gl3wStat = gl3wInit();
60          if (gl3wStat != 0)
61          {
62              Err("Failed to initialize gl3w!");
63          }
64
65          SDL_GL_SetSwapInterval(vsync);
66
67          this->state = State::RUNNING;
68
69          last = SDL_GetTicks();
70          Init();
71
72          SDL_version compiled;
73          SDL_VERSION(&compiled)
74
75          std::wcout << "Skeletal Animation Engine V4"<<
        (wchar_t)0xA9 <<" Alpha Version" << std::endl;
76          printf("Build with: ");
77          std::cout << "OpenGL Version: " <<
        glGetString(GL_VERSION) << std::endl;
78          std::cout << "GLSL Version: " <<
        glGetString(GL_SHADING_LANGUAGE_VERSION) << std::endl;
79          printf("SDL Version: %d.%d.%d \n\n", compiled.major,
        compiled.minor, compiled.patch);
80
81          //Own Engine Loop
82          while (State::RUNNING == state)
83          {
84              float deltaTime = GetDeltaTime();
85              GetFPS();
86              /*PollInput();*/
87              Input();
88              Update(deltaTime);
89              Render();
90              SDL_GL_SwapWindow(window);
91              LimitFPS();
92              PrintFPS();
93          }
94          DeInit();
95      }
96
97      //Destroy Core Object
98-107 void Core::DeInit(){...}
108
109     //Build Shader
110-192 GLuint Core::BuildShader(const char * vertexPath, const char *
193     fragmentPath, const char * geometryPath){...}
```

```
194
195     //Use Shader
196     void Core::UseShader(GLuint program)
197     {
198             // Uses the current shader
199             glUseProgram(program);
200     }
201
202     //Get Delta Time
203     float Core::GetDeltaTime()
204     {
205             unsigned int time = SDL_GetTicks();
206             unsigned int delta = time - lastFrame;
207             lastFrame = time;
208
209             return delta;
210     }
211
212     //Get FPS
213     void Core::GetFPS()
214     {
215             if (SDL_GetTicks() - last > 1000) {
216                     fps = _fps;
217                     _fps = 0;
218                     last += 1000;
219             }
220             _fps++;
221     }
222
223-243  //Input Poll
244     void Core::PollInput(){...}
245
246
247     //Error Check
248     void Core::Err(string errorString)
249     {
250             cout << errorString << endl;
251             SDL_Quit();
252             exit(1);
253     }
254-258
259     //FPS Limitter
260     void Core::LimitFPS(){...}
261-283
284
285     //Check if Any Shader Error
286-295  void Core::CheckShaderErrors(GLuint shader, string type)
         {...}

         //Print FPS
         void Core::PrintFPS(){...}
```

A.2 *Class AnimationEngine*

A.2.1 AnimationEngine.h

No	AnimationEngine.h
1	#pragma once
2	#include "Core.h"
3	#include "Component2D.h"
4	#include "Joint2D.h"
5	#include "Frame.h"
6	#include "Bone.h"
7	

```
8   enum class KinectStat { ACTIVE, DEACTIVE };
9   enum class SaveMode { OVERWRITE, SAVE_EXISTING };
10
11  class AnimationEngine :
12          public Core
13  {
14  public:
15          AnimationEngine();
16          ~AnimationEngine();
17          virtual void Init();
18          virtual void Render();
19          virtual void Update(float deltaTime);
20          virtual void Input();
21
22  private:
23          void initKinect();
                    //Initialize Kinect
24          void deactivateKinect();
            //Deactivate Kinect
25          void getAllKinectData();
            //Get All Data Stream From Kinect
26          void toggleKinect(KinectStat stat);
            //Toggle Kinect
27          void getBodyData(IMultiSourceFrame* frame);
            //Get Body Data From Stream
28          void AnimateFrame(vector<Frame> a);
            //Animating Keyframe
29          void saveCoordinateInNote(SaveMode savemode);          //Save
    Coordinate To .TXT File
30          void saveFrameInVector();
            //Save Frame in Vector //Like Array List
            void displaySingleFrame(vector<Frame> a);
31          //Display Single Frame
            void readCoordinate(const char * fileName);
32          //Read Coordinate From .TXT File
            void playImportKeyframe(bool play);
33          //Play Imported Keyframe
            void playExistingKeyframe(bool play);
34          //Play Directly
            void renderComponen();
35                  //Render Componen
            void initGui();
36                          //Initialize GUI
            void makeGui();
37                          //Make GUI
            void renderGui();
38                  //Render GUI
            GLfloat angleFromArbitraryOrigin(vec4 a, vec4 b, vec4
39  aUpdate, vec4 bUpdate);    //Method to get Angle From Other Origin

            //User define object
40          Component2D kepala, tubuh, kaki_kiri, kaki_kanan,
41  tangan_kiri, tangan_kanan;
42          Joint2D JointFromKinect[JointDirectory_Count];
            Joint2D JointForAnimation[JointDirectory_Count];
43          Joint2D JointForSingleFrame[JointDirectory_Count];
44          Joint2D JointFromTxt[JointDirectory_Count];
45          Bone ConnectingBone[JointDirectory_Count - 1];
46          Frame frames, keyframesFromText;
47          string path;
48          KinectStat status;
49
50          //Make Vector Frame
51          vector<Frame> ExistingKeyframe;
52          vector<Frame> keyframeFromText;
```

```
53
54          int totalVector = 1;
55
56          vec4 lerpJoint[JointDirectory_Count];
57          vec4 temporary[JointDirectory_Count];
58          vec4 resultInterpolate[JointDirectory_Count];
59
60          vec2 vectorAtoB_before[4];
61          vec2 vectorAtoB_after[4];
62
63          GLfloat lengthVector[4];
64
65          GLfloat angleVector[8];
66  };
67
68
```

A.2.2 AnimationEngine.cpp

No	AnimationEngine.cpp
1	#include "AnimationEngine.h"
2	
3	#pragma region Kinect Variable
4	IKinectSensor* sensor;
5	IMultiSourceFrameReader* multiSourceReader;
6	ICoordinateMapper* mapper;
7	Joint joints[21];
8	ColorSpacePoint colorPoint;
9	CameraSpacePoint cameraCoordinate;
10	CameraSpacePoint csc[JointDirectory_Count];
11	ColorSpacePoint csp[JointDirectory_Count];
12	BOOLEAN tracked;
13	#pragma endregion
14	
15	static bool playImportedKeyframe = false;
16	static bool playExisting = false;
17	static bool renderHead = false;
18	static bool renderTorso = false;
19	static bool renderLeftLeg = false;
20	static bool renderRightLeg = false;
21	static bool renderLeftArm = false;
22	static bool renderRightArm = false;
23	static bool renderJoint = false;
24	
25	static float f_head = 0.5f;
26	static float f_torso = 0.5f;
27	static float f_right_arm = 0.5f;
28	static float f_left_arm = 0.5f;
29	static float f_right_leg = 0.5f;
30	static float f_left_leg = 0.5f;
31	
32	#pragma region All Function
33-	AnimationEngine::AnimationEngine(){...}
35	
36	
37-	AnimationEngine::~AnimationEngine(){...}
39	
40	
41	//Initialize
42-	void AnimationEngine::Init(){...}
87	
88	
89	//Rendering
90	void AnimationEngine::Render()
91	{

```
92          makeGui();
93
94          renderGui();
95
96          SDL_GL_MakeCurrent(window, glContext);
97
98          //Setting Viewport
99          glViewport(0, 0, width, height);
100         //Clear the color and depth buffer
101         glClear(GL_COLOR_BUFFER_BIT);
102
103         //Set the background color
104         glClearColor(0.8f, 0.8f, 0.8f, 1.0f);
105
106         //Render Component
107         renderComponen();
108
109         toggleKinect(status);
110
111         playImportKeyframe(playImportedKeyframe);
112
113         playExistingKeyframe(playExisting);
114
115         ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
    }
116
117 //Update
118 void AnimationEngine::Update(float deltaTime){...}
119-
122 //The Keyboard Controller
123 void AnimationEngine::Input(){...}
124-
161
162 //Activate Kinect
163 void AnimationEngine::initKinect()
164 {
165         GetDefaultKinectSensor(&sensor);
166         if (sensor != NULL)
167         {
168                 sensor->Open();
169                 sensor->get_CoordinateMapper(&mapper);
170                 sensor->OpenMultiSourceFrameReader(
171                         FrameSourceTypes_Color |
172                         FrameSourceTypes_Body,
173                         &multiSourceReader
174                 );
175         }
176         else
177         {
178                 printf("Kinect Not Available!\n");
179         }
180 }
181
182 //Deactivate Kinect
183 void AnimationEngine::deactivateKinect()
184 {
185         sensor->Close();
186 }
187
188 //Get All Kinect Data Stream
189 void AnimationEngine::getAllKinectData(){...}
190-
197
198 //Toggle Kinect
199 void AnimationEngine::toggleKinect(KinectStat stat){...}
```

```
200-
212
213   //Get Skeleton Data from Kinect
214   void AnimationEngine::getBodyData(IMultiSourceFrame *
215-  frame){...}
309
310   //Animating Frame
311   void AnimationEngine::AnimateFrame(vector<Frame> a)
312   {
313         deactivateKinect();
314
315         int vectorSize = a.size();
316         int ProsesLerp = vectorSize - 1;
317
318         static int n = 0;
319
320         if (n < ProsesLerp)
321         {
322               Joint2D * temp = a[n].getFrame();
323               Joint2D * temp2 = a[n + 1].getFrame();
324
325               static float lerpValue = 0.00f;
326               lerpValue += 0.01f;
327               if (lerpValue < 1.00f)
328               {
329                     //Clear the color and depth buffer
330                     glClear(GL_COLOR_BUFFER_BIT);
331
332                     for (int j = 0; j < JointDirectory_Count;
333   j++)
                      {
334                           lerpJoint[j] =
335   mix(temp[j].getJointVector(), temp2[j].getJointVector(),
      lerpValue);
                           if (renderJoint)
336                           {
337
      JointForAnimation[j].drawJoint(lerpJoint[j]);
338                           }
339                     }
340                     renderComponen();
341
342                     ////Draw Component at The Origin
343                     //Head
344                     {
345                           kepala.setPos(lerpJoint[0].x + 15.0f,
346   lerpJoint[0].y);
                            kepala.setSize(vec4(f_head, f_head,
347   0.0f, 0.0f));
                      }
348
349                     //Torso
350                     {
351                           tubuh.setPos(((lerpJoint[1].x +
352   lerpJoint[8].x) / 2), ((lerpJoint[1].y + lerpJoint[8].y) / 2) -
      44.5f);
                           tubuh.setSize(vec4(f_torso, f_torso,
353   0.0f, 0.0f));
                      }
354
355                     //Right_arm
356                     {
357                           GLfloat rotateAtShoulder =
358   angleFromArbitraryOrigin(temp[5].getJointVector(),
      temp[6].getJointVector(), lerpJoint[5], lerpJoint[6]);
```

```
                                tangan_kanan.setPos(lerpJoint[6].x,
    lerpJoint[6].y);
359                             tangan_kanan.rotateComponen(-
360 rotateAtShoulder + (-90.0f));
                                tangan_kanan.setSize(vec4(f_right_arm,
    f_right_arm, 0.0f, 0.0f));
361                         }
362
363                         //Left_arm
364                         {
365                             tangan_kiri.setPos(0.5f * width, 0.5f
366 * height);
                                /*lerpJoint[3].x, lerpJoint[3].y*/
367                             tangan_kiri.setSize(vec4(f_left_arm,
    f_left_arm, 0.0f, 0.0f));
368                         }

369                         //Right_leg
370                         {
371                             kaki_kanan.setPos(lerpJoint[13].x,
372 lerpJoint[13].y);
373                             kaki_kanan.setSize(vec4(f_right_leg,
    f_right_leg, 0.0f, 0.0f));
                            }
374
                        //Left_leg
375                     {
376                         kaki_kiri.setPos(lerpJoint[10].x,
377 lerpJoint[10].y);
378                         kaki_kiri.setSize(vec4(f_left_leg,
379 f_left_leg, 0.0f, 0.0f));
                        }
380
                    if (lerpValue > 0.99f && lerpValue < 1.0f)
381                 {
382                     n = n + 1;
383                     lerpValue = 0.00f;
                    }
384             }
385     }
386 }
387
388 //Saving coordinate in notepad using ofstream
389 void AnimationEngine::saveCoordinateInNote(SaveMode
390 savemode){...}
391
392 //Saving coordinate in vector (Like dynamic array)
393- void AnimationEngine::saveFrameInVector()
432 {
433     //Simpan dengan vector
434     frames = Frame(JointFromKinect);
435     ExistingKeyframe.push_back(frames);
436     cout << "Vector Frame : " << totalVector++ << endl;
437 }
438
439 //Display Single Frame
440 void AnimationEngine::displaySingleFrame(vector<Frame> a)
441 {...}
442
443 //Read Coordinate of Joint from .txt file
444- void AnimationEngine::readCoordinate(const char * fileName){...}
461
462 //Play Animation From Imported Keyframe
463 void AnimationEngine::playImportKeyframe(bool play)
    {
```

```
464-        if (play)
494         {
495             AnimateFrame(keyframeFromText);
496         }
497         else {
498             status = KinectStat::ACTIVE;
499         }
500 }
501
502 //Play Animation Directly
503 void AnimationEngine::playExistingKeyframe(bool play)
504 {
505         if (play)
506         {
507             AnimateFrame(ExistingKeyframe);
508         }
509         else {
510             status = KinectStat::ACTIVE;
511         }
512 }
513
514 //Rendering Component
515 void AnimationEngine::renderComponen(){...}
516
517
518 //Initialize GUI
519 void AnimationEngine::initGui(){...}
520
521-
553 //Make GUI Here <<---- ADD HERE
554 void AnimationEngine::makeGui(){...}
555
556-
574 //Render GUI
575 void AnimationEngine::renderGui()
576 {
577-        // Rendering
784         ImGui::Render();
785 }
786
787 //Rotate Arbitrary Origin
788 GLfloat AnimationEngine::angleFromArbitraryOrigin(vec4 a, vec4
789 b, vec4 aUpdate, vec4 bUpdate){...}
790
791 #pragma endregion
792
793
794-
830
831
832
```

A.3 *Class Component*

A.3.1 Component2D.h

No	Component2D.h
1	#pragma once
2	#include "Core.h"
3	class Component2D :
4	public Core
5	{
6	public:
7	Component2D();

```
8           ~Component2D();
9           void drawComponen();
10          void initComponen(string ComponenName);
11          void setSize(vec4 size);
12          void setPos(GLfloat x, GLfloat y);
13          void setPos(ColorSpacePoint & csp);
14          void setPosCustomOrigin(GLfloat x, GLfloat y);
15          void rotateComponen(GLfloat degree);
16          int getWidthScreen();
17          int getHeightScreen();
18          virtual void Init();
19          virtual void Render();
20          virtual void Update(float deltaTime);
21          virtual void Input();
22          virtual void DeInit();
23
24  private:
25          GLuint VBO, VAO, EBO, texture, program;
26          mat4 model, view, projection;
27  };
```

A.3.2 Component2D.cpp

```
No      Component2D.cpp
1       #include "Component2D.h"
2
3-5     Component2D::Component2D(){...}
6
7-9     Component2D::~Component2D(){...}
10      //Render 2D Sprite
11-     void Component2D::drawComponen(){...}
32
33      //Initialize 2D Sprite
34-     void Component2D::initComponen(string ComponenName){...}
96
97      //Set Size
98-     void Component2D::setSize(vec4 size){...}
103
104     //Set Position
105-    void Component2D::setPos(GLfloat x, GLfloat y){...}
111
112     //Set Position
113-    void Component2D::setPos(ColorSpacePoint & csp){...}
119
120     //Set Custom Origin
121-    void Component2D::setPosCustomOrigin(GLfloat x, GLfloat y){...}
126     //Rotate Componen
127     void Component2D::rotateComponen(GLfloat degree){...}
128-
133     //Get Width of Screen
134     int Component2D::getWidthScreen()
135     {
136             return width;
137     }
138     //Get Height of Screen
139     int Component2D::getHeightScreen()
140     {
141             return height;
142     }
143     // ================================================
144     // Derived Method
145     void Component2D::Init(){...}
146-
148
149     void Component2D::Render(){...}
```

```
150-
152
153   void Component2D::Update(float deltaTime){...}
154-
156
157   void Component2D::Input(){...}
158-
160
161   void Component2D::DeInit(){...}
162
```

A.4 *Class Joint2D*

A.4.1 Joint2D.h

```
No  Joint2D.h
1   #pragma once
2   #include "Core.h"
3   class Joint2D :
4         public Core
5   {
6   public:
7         Joint2D();
8         ~Joint2D();
9         void initJoint();
10        void setJointVector(GLfloat x, GLfloat y);
11        void drawJoint(ColorSpacePoint & colorPoint);
12        void drawJoint(vec4 vectorPerJoint);
13        void updateJointPos(vec4 vectorPerJoint);
14        vec4 getJointVector();
15        mat4 getModelMatrix();
16
17        virtual void Init();
18        virtual void Render();
19        virtual void Update(float deltaTime);
20        virtual void Input();
21
22  private:
23        GLuint VBO, VAO, EBO, program;
24        vec4 vectorJoint;
25        mat4 model, view, projection;
26  };
27
28  enum jointDirectory
29  {
30        Head = 0,
31        Neck = 1,
32        ShoulderLeft = 2,
33        ElbowLeft = 3,
34        HandLeft = 4,
35        ShoulderRight = 5,
36        ElbowRight = 6,
37        HandRight = 7,
38        SpineBase = 8,
39        HipLeft = 9,
40        KneeLeft = 10,
41        AnkleLeft = 11,
42        HipRight = 12,
43        KneeRight = 13,
44        AnkleRight = 14,
45        JointDirectory_Count = (AnkleRight + 1)
46  };
```

A.4.2 Joint2D.cpp

No	Joint2D.cpp
1	#include "Joint2D.h"
2	
3-5	Joint2D::Joint2D(){...}
6	
7-9	Joint2D::~Joint2D(){...}
10	//Initialize Joint
11-	void Joint2D::initJoint(){...}
46	
47	//Set Joint Vector
48-	void Joint2D::setJointVector(GLfloat x, GLfloat y){...}
54	
55	//Render Joint
56-	void Joint2D::drawJoint(ColorSpacePoint & colorPoint){...}
67	
68	//Render Joint
69-	void Joint2D::drawJoint(vec4 vectorPerJoint){...}
80	
81	//Update Joint Position
82-	void Joint2D::updateJointPos(vec4 vectorPerJoint){...}
87	
88	//Get Joint Vector
89	vec4 Joint2D::getJointVector()
90	{
91	return vectorJoint;
92	}
93	//Get Model Matrix
94	mat4 Joint2D::getModelMatrix()
95	{
96	return model;
97	}
98	// ====================================
99	// Derived Method
100-	void Joint2D::Init(){...}
102	
103	
104-	void Joint2D::Render(){...}
106	
107	
108-	void Joint2D::Update(float deltaTime){...}
110	
111	
112-	void Joint2D::Input(){...}
114	

A.5 *Class Frame*

A.5.1 Frame.h

No	Frame.cpp
1	#include "Frame.h"
2	
3	Frame::Frame()
4	{
5	}
6	
7	Frame::Frame(Joint2D body[])
8	{
9	for (int i = 0; i < JointDirectory_Count; i++)
10	{
11	this->JointInSkeleton[i] = body[i];
12	}

```
13  }
14
15  Frame::~Frame()
16  {
17  }
18
19  Joint2D * Frame::getFrame()
20  {
21        return JointInSkeleton;
22  }
```

A.5.2 Frame.cpp

No	Frame.h
1	#pragma once
2	#include "Joint2D.h"
3	class Frame
4	{
5	public:
6	Frame();
7	Frame(Joint2D body[]);
8	~Frame();
9	
10	Joint2D * getFrame();
11	private:
12	Joint2D JointInSkeleton[JointDirectory_Count];
13	};