**BAB III**

**PERANCANGAN DAN IMPLEMENTASI**

**3.1 Gambaran Umum**

Aplikasi yang dibuat dalam penulisan ini mampu menampilkan animasi sebuah figur *skeleton* beserta dengan *body* *center of mass* dan *segments center of mass* dalam dunia tiga dimensi interaktif berupa BVH *player* dengan seratus gambar per detik menggunakan. Aplikasi ini menyediakan sebuah kamera yang dapat diarahkan dengan menggunakan *mouse* dapat digerakkan dengan menggunakan *keyboard*. Aplikasi ini memiliki berbagai pengaturan warna dan kriteria yang dapat diubah untuk memudahkan proses pengamatan.

Gerakan animasi yang ditampilkan didapatkan dari membaca sebuah *file* berekstensi BVH. Sebuah *file* BVHberisi kumpulan data pergerakan tubuh manusia yang telah direkam dan diterjemahkan kedalam bentuk teks dengan pola tertentu. Data dari teks tersebut di-*parsing* kedalam bentuk *tree* sebagai struktur data *skeleton* dalam melakukan proses *rendering*. Struktur data *tree* yang telah dimanipulasi sedemikian rupa kemudian diubah kedalam bentuk *linked list* untuk melakukan proses penghitungan posisi dari setiap *joints, segments* COM*,* dan *body* COM. Kalkulasi dari masing-masing COM dilakukan berdasarkan persamaan-persamaan yang ditampilkan pada bab sebelumnya. Aplikasi kemudian melakukan *rendering* dari setiap *joints, segments* COM*,* dan *body* COM dengan posisi terbaru. Proses ini dilakukan berulang-ulang dalam interval satu *millisecond* sampai *frame* terakhir yang dinamakan *update loop*.

Aplikasi dapat melakukan *plotting* setiap *segments* COMdan *body* COMkedalam bentuk *histogram*. Penentuan elemen yang akan dilakukan *plotting* diseleksi secara manual. Hasil dari *plotting* berisi posisi setiap COM terhadap *frame*. Setiap hasil *plotting* akan hilang setelah aplikasi ditutup.

**3.2 Kerangka Penelitian**

Kerangka penelitian yang jelas dibutuhkan untuk memudahkan proses pembuatan aplikasi ini dan dapat terencana dengan baik sehingga mempersingkat waktu pengerjaan. Proses pembuatan aplikasi ini dibagi menjadi tiga tahapan, yaitu tahap pra produksi, tahap produksi, dan tahap uji coba. Setiap langkah dilakukan secara terurut. Uraian setiap tahap dapat dilihat pada gambar 3.1.

 **Gambar 3.1 Kerangka Penelitan**

**3.3 Tahap Pra Produksi**

Tahap pra produksi berisi langkah-langkah analisis yang menentukan alur pada tahapan selanjutnya. Tahap pra produksi dibagi menjadi beberapa aktivitas yaitu analisis kebutuhan aplikasi, analisis struktur projek, dan analisis data.

**3.3.1 Analisis Kebutuhan Aplikasi**

Pembuatan aplikasi ini memerlukan alat-alat pendukung berupa perangkat keras (*hardware*) dan perangkat lunak (*software*). Spesifikasi perangkat keras yang digunakan dalam penulisan ini dapat dilihat pada tabel 3.1, sedangkan spesifikasi perangkat lunak yang digunakan dapat dilihat pada tabel 3.2.

**Tabel 3.1 Spesifikasi Perangkat Keras**

|  |  |
| --- | --- |
| **Perangkat Keras (Laptop)** | |
| CPU | Intel Core I7 7700 HQ |
| GPU | NVIDIA GTX 1060 6GB |
| RAM | 24 GB DDR4 |
| SSD | NVME SAMSUNG 120 GB |
| HDD | SATA 1 TB |

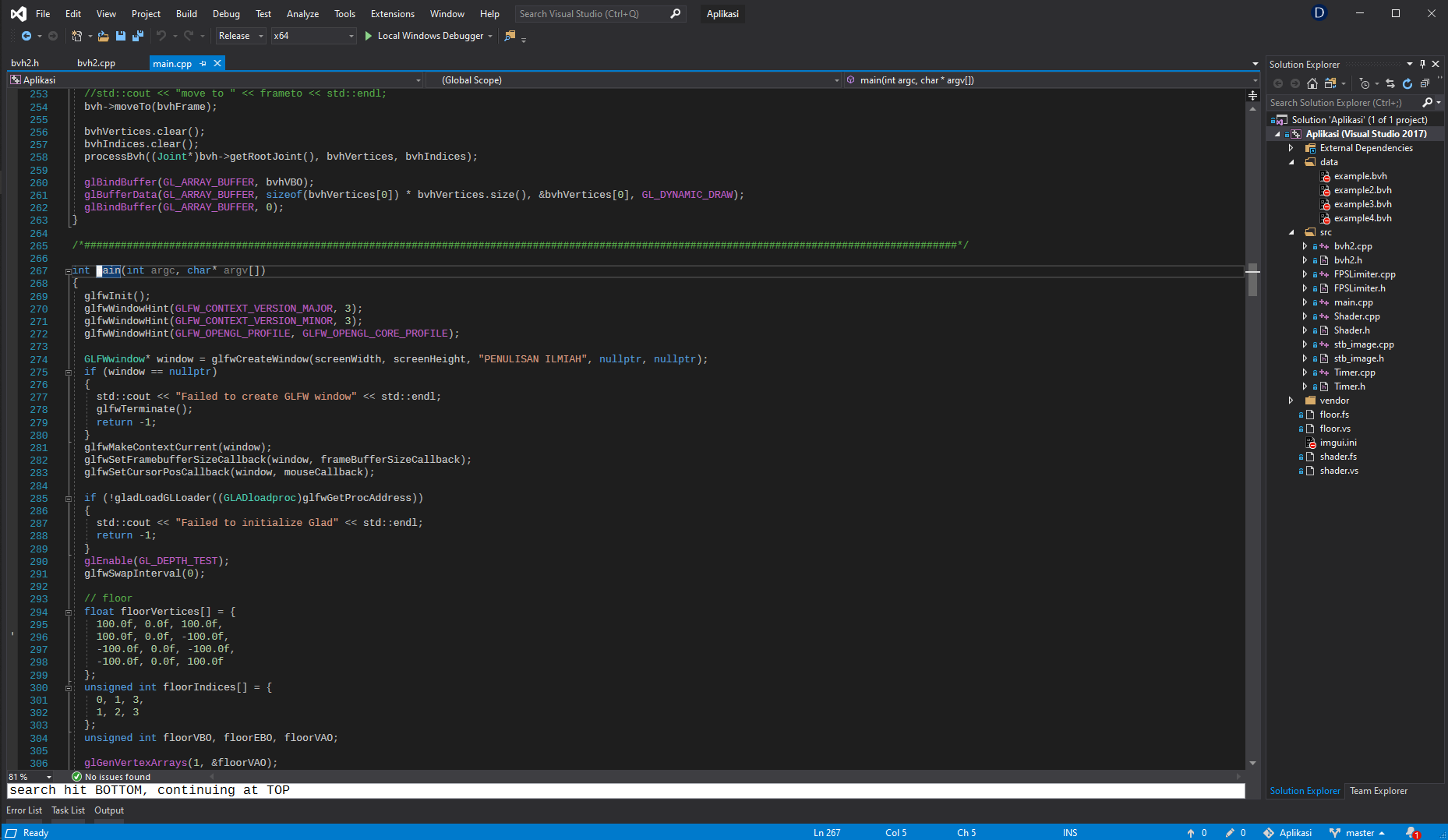
**Tabel 3.2 Spesifikasi Perangkat Lunak**

|  |  |
| --- | --- |
| **Perangkat Lunak** | |
| Sistem Operasi | Microsoft Windows 10 Home Edition |
| IDE | Visual Studio 15 2017 Community Edition |
| Project Generator | Premake 5 |
| Compiler | Microsoft Visual C++ Compiler |
| Internet Browser | Mozilla Firefox Quantum |
| Version Control Client | Git |
| Version Control Hosting | GitHub |
| Terminal | Cmder dan Windows Terminal |

**3.3.2 Analisis Struktur Projek**

Perancangan struktur projek yang baik dapat meminimalkan tingkat kompleksitas dalam mengerjakan suatu projek. Tingkat kompleksitas yang rendah mempermudah pengerjaan pada tahap-tahap selanjutnya.

Perancangan struktur projek bersifat *platform dependent* dan *in-source*. Projek yang bersifat *platform dependent* berarti hasil dari projek ini hanya dapat dibuat dan dijalankan pada satu sistem operasi saja. Isi dari *source code* yang dibuat dalam projek ini tidak dapat digunakan pada sistem operasi lain. Pengerjaan aplikasi ini menggunakan Visual Studio 15 2017 Community Edition pada sistem operasi Windows 10 Home. Contoh penggunaannya ditampilkan pada gambar 3.2.



**Gambar 3.2 Visual Studio 2017 Community Edition**

Pengaplikasian projek yang bersifat *in-source* yang berarti berkas-berkas dikategorikan menjadi beberapa direktori berdasarkan jenis dan fungsinya. *Source code* yang diketik dipisahkan dengan *file executable* beserta dengan data yang akan dipakai. Pembagian direktori dilakukan dengan kostumisasi projek Visual Studio dengan Premake*5*.

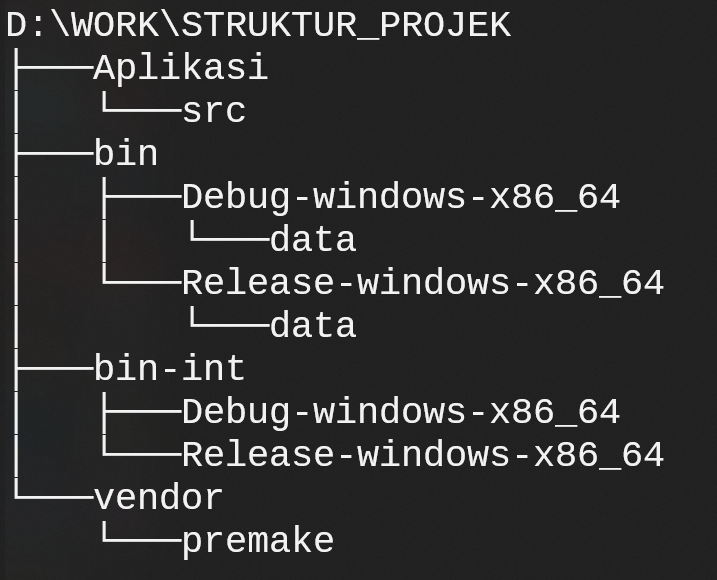
Direktori root (STRUKTUR\_PROJEK) merupakan akar teratas dari subdirektori lainnya. Aplikasi adalah direktori projek yang dalam hal ini merupakan nama projek yang dikerjakan. Di dalamnya terdapat direktori src yang berisi semua *source code* yang akan digunakan.

Direktori bin (*binary*) berisi 2 subdirektori yang masing-masing berisi *debug* *executable* dan *release executable*. *Debug executable* merupakan program yang didalamnya disisipkan simbol-simbol *debug* sehingga memudahkan pencarian *bug*. *Release executable* adalah program yang sama dengan *debug* tetapi sudah dioptimisasi dengan baik sehingga siap untuk dipakai.

Direktori bin-int (*binary-intermediate*) berisi berkas-berkas hasil kompilasi. Microsoft Visual C++ Compiler (MSVC)mengubah *source code* yang telah diketik kedalam *file* obj yang berisi representasi bahasa mesin. Setiap *file* obj dikelompokkan kedalam *debug* dan *release* yang pada akhirnya akan dilakukan *link* oleh *linker* dan disimpan dalam direktori bin.

Direktori vendor menampung perangkat lunak yang membantu proses pengerjaan aplikasi yang meliputi *tools* dan *library.* Penulisan ini menggunakan premake5 sebagai alat bantu pembentukan projek, sehingga berkas-berkas yang berkaitan dengan premake disimpan didalam direktori vendor.

Penyesuaian projek seperti yang telah dijelaskan dengan rinci memudahkan pembuatan aplikasi . Penggambaran direktori projek secara mendalam diilustrasikan dalam bentuk *tree* seperti pada gambar 3.3.



**Gambar 3.3 Struktur Projek**

**3.3.3 Analisis Data**

Data yang dipakai sebagai masukan aplikasi merupakan informasi pergerakan dari perekaman alat *motion capture* dalam format animasi *Biovision Hierarchy* (BVH). Sebuah *file* BVHdibagi menjadi dua bagian besar yaitu bagian *hierarchy* dan *motion*. Bagian pertama yaitu bagian hierarki akan dikerjakan terlebih dahulu kemudian diikuti dengan bagian pergerakan.

1. Hierarchy
2. *Root* mendefinisikan sendi (pinggul) yang merupakan awal dari hierarki.
3. *Joint* dapat mendefinisikan suatu penulangan yang dibutuhkan seperti tulang belakang, tungkai, lengan, telapak tangan, dan sebagainya.
4. *End Site* merupakan akhir dari suatu penulangan yang hanya menjadi representasi visual.
5. *Offset* adalah posisi relatif suatu *joint* terhadap *parent*-nya pada setiap sumbu.
6. *Channels* adalah jenis transformasi yang akan dipakai pada *joint* atau *root* terkait. Transformasi tersebut dapat berupa translasi dan atau rotasi.
7. Motion
8. *Frames* ialah jumlah *frame* yang terekam.
9. *Frame Time* adalah selisih waktu antara suatu *frame* dengan yang lainnya dalam satuan *millisecond*. Penentuan *fps* dapat dihitung berdasarkan *frame time*.
10. *Data* merupakan informasi transformasi setiap *channel* yang terkait dengan masing-masing *joint*-nya yang terurut sesuatu dengan pendefinisian hierarki.

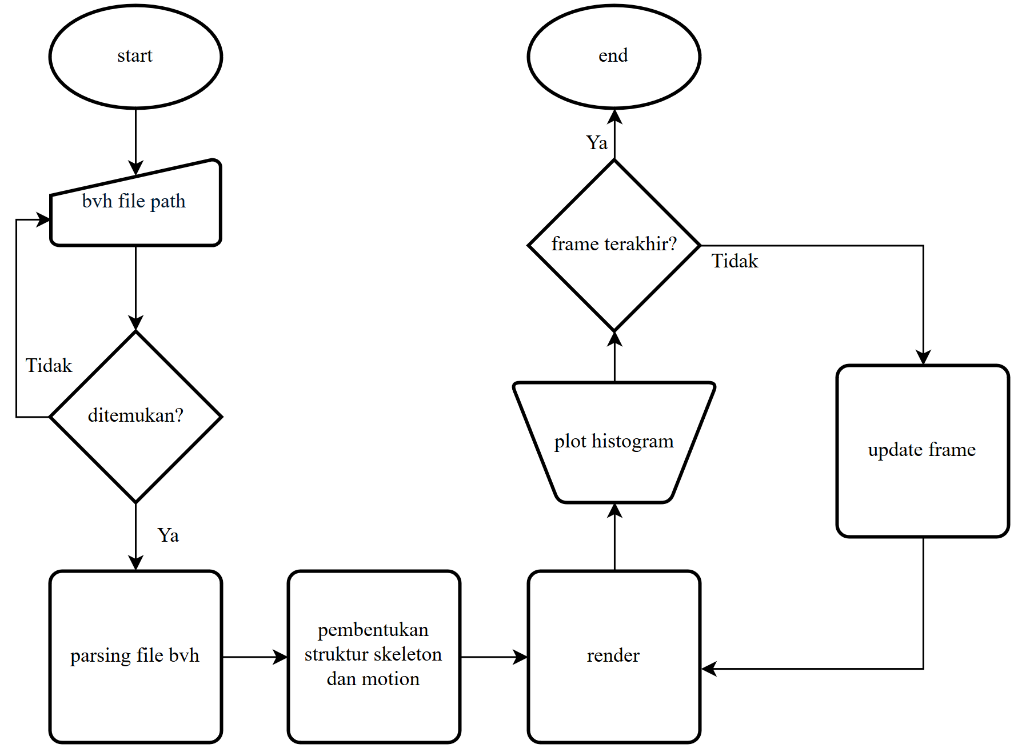
**3.4 Tahap Produksi**

Tahap kedua adalah tahap produksi dimana pengerjaan program dimulai. Tahap ini terdiri dari lima aktivitas utama, yaitu penentuan alur program, proses *parsing file* BVH*,* kalkulasi pusat massa, *rendering*, dan *plotting histogram*.

**3.4.1 Alur Program**

Aplikasi yang akan dirancang merupakan program berbasis objek yang direpresentasikan dalam *flowchart diagram*. *Flowchart diagram* digunakan untuk menjelaskan alur aplikasi secara umum. Perancangan aplikasi dibagi menjadi empat bagian besar, yaitu *parsing file* BVH, *rendering* menggunakan OpenGL, dan *plotting histogram*. Alur program secara garis besar digambarkan pada gambar 3.4 meliputi:

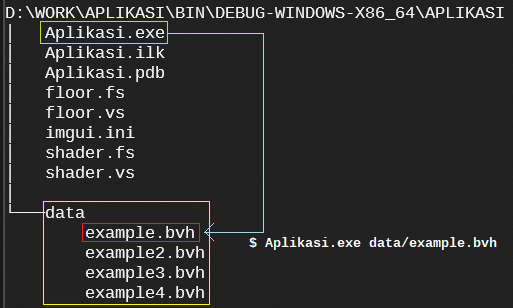
1. Hal pertama yang dilakukan oleh program setelah mendapat masukan nama *file* BVH yang berbentuk adalah memverifikasi apakah teks tersebut benar-benar mewakili suatu file di dalam direktori.
2. Apabila langkah verifikasi ini gagal, maka program akan meminta ulang pemasukkan teks yang valid. Sebaliknya apabila verifikasi berhasil, maka program akan mencoba melakukan *parsing* *file* tersebut.
3. Kemudian program akan membentuk sebuah *tree* bernama “BVH\_hierarchy” dan sebuah *array* bernama “BVH\_motion”. BVH\_hierarchy berisi informasi mengenai struktur *skeleton* yang akan di-*render*, sedangkan BVH\_motion berisi informasi animasi pergerakan setiap elemen dari BVH\_hierarchy pada tiap-tiap *frame*.
4. Program kemudian melakukan *rendering* sesuai dengan pilihan warna dan kamera yang diatur.
5. Dilanjutkan dengan *plotting histogram* terhadap COM pilihan secara manual.
6. Selanjutnya program akan memeriksa apakah angka *frame* yang sedang digunakan merupakan *frame* terakhir. Jika *frame* masih bukan merupakan *frame* terakhir, maka akan dilakukan *update skeleton* terhadap data *motion* yang kemudian diikuti dengan proses *render* (d) dan *plotting* (e).



**Gambar 3.4 Alur Program**

**3.4.2 Parsing File BVH**

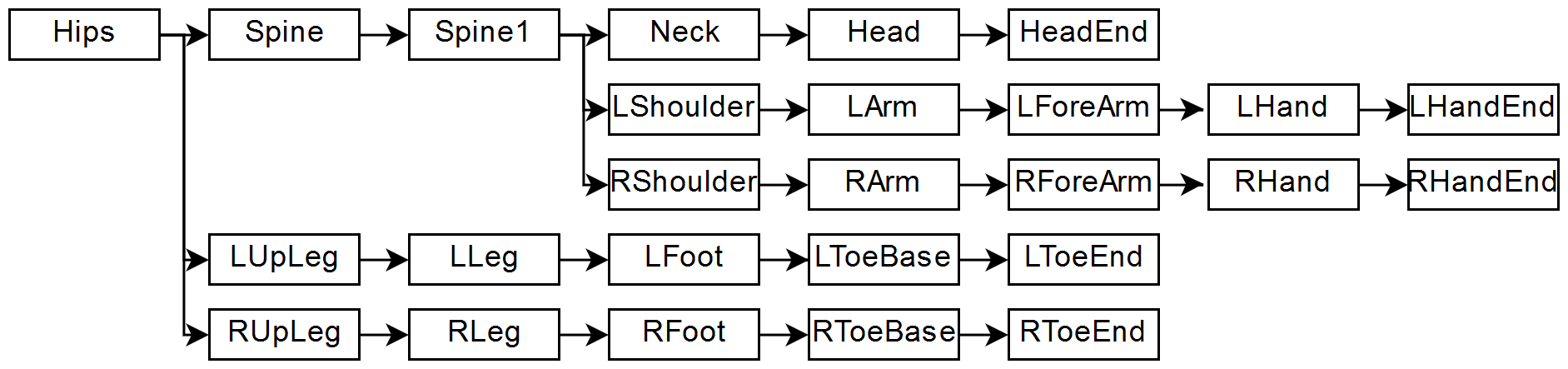
Program dimulai dengan nama *file* BVH yang akan diamati secara manual. Cara melakukan *input* ialah melalui *command line argument*. *Command line argument* merupakan sebuah atau sejumlah kata kunci yang ingin diberikan kepada suatu program menggunakan perintah-perintah tertentu melalui sebuah konsol atau *terminal*. Skema dari *command line argument* diilustrasikan pada gambar 3.5.



***Gambar 3.5 Skema Passing Command Line Argument***

Program akan mencoba membaca isi dari berkas tersebut. Cara pembacaan dilakukan dengan menggunakan kelasfstreamyang disediakan oleh C++ Standard Template Library. Kelasini berguna untuk membaca berkas dan dilengkapi dengan beberapa *method*.

Sebuah objek dengan tipe data BVH2 bernama BVH diinisialisasi sesaat sesudah memulai aplikasi. Objek ini berisi struktur *skeleton,* informasi animasi pergerakan, jumlah *frame*, *frame time*, dan nama-nama *joint* yang telah disusun sedemikian rupa untuk mempermudah pembacaan. Objek BVH memiliki *method* loadHierarchy(), loadJoint(), dan loadMotion() yang berguna untuk membaca *file* BVH. Struktur skeleton yang dibuat diilustrasikan pada gambar 3.6.

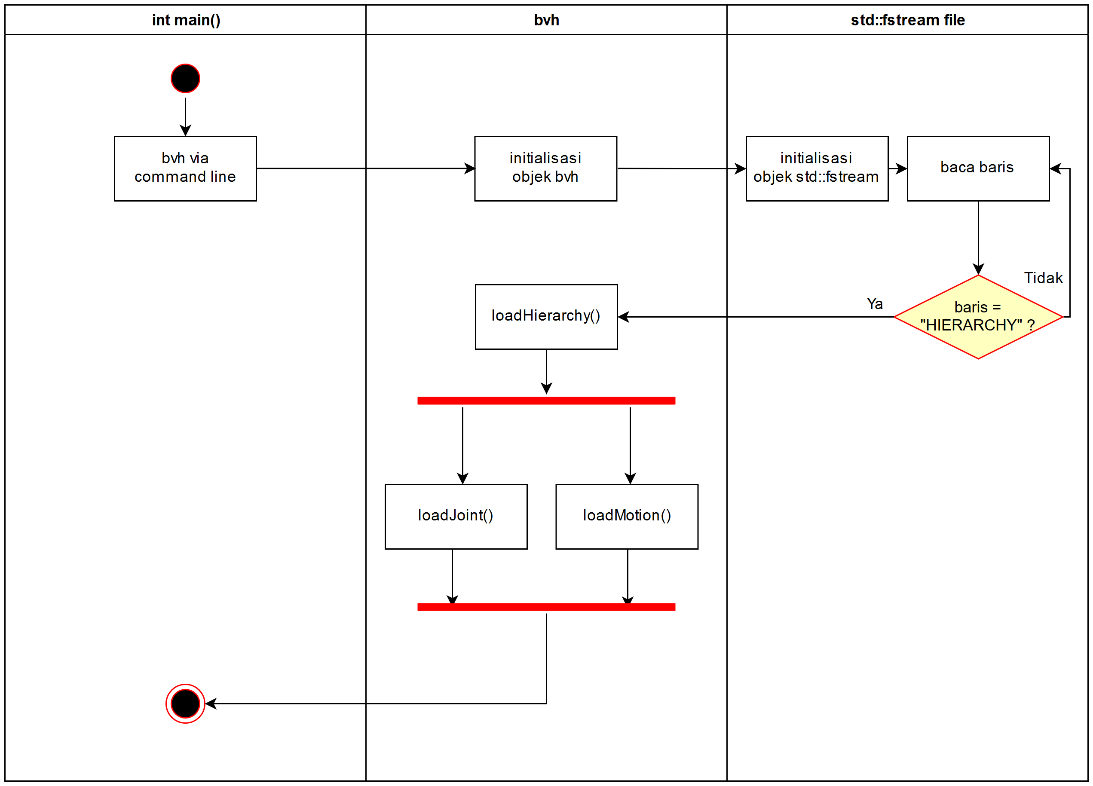


***Gambar 3.6 Struktur Data Skeleton***

Metode yang dipakai untuk membaca file adalah dengan membaca file secara baris per baris. Hal yang dilakukan adalah membuat sebuah objek fstreambernama “file*”*. Objek ini kemudian akan dibaca baris per baris. Setiap baris yang terbaca akan dilakukan validasi mengenai apakah baris tersebut berisi kalimat “HIERACHY”. Objek tersebut akan dibaca struktur akan diteruskan ke *method* loadHierarchy(std::istream& stream).

Setelah objek berada di *method* loadHierarchy(std::istream& stream), program akan melakukan pemeriksaan apakah baris yang dibaca berisi kata “ROOT” atau “MOTION”. Percabangan ini akan melanjutkan objek ke *method* loadJoint(std::istream& stream) bila kata yang ditemukan adalah “ROOT”. *Method* ini berfungsi untuk membangun data *skeleton* dalam bentuk *tree*. Langkah selanjutnya adalah mencari kata “MOTION” dan jika ditemukan maka objek tersebut akan diberikan kepada *method* loadMotion(std::istream& stream) yang akan membuat suatu *array* berisi informasi transformasi dari setiap *joint* pada *skeleton*.

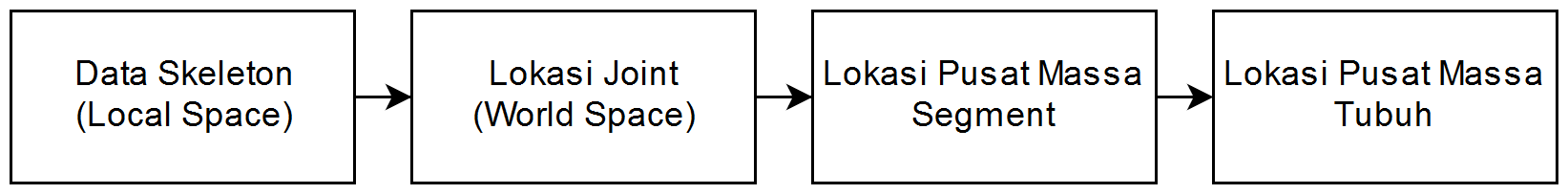
Kedua data ini kemudian dilanjutkan pada langkah selanjutnya yaitu proses *rendering* menggunakan *modern* OpenGL. Alur dari proses melakukan *parsing* diilustrasikan pada gambar 3.7.



***Gambar 3.7 Skema Parsing* BVH**

**3.4.3 Kalkulasi Center of Mass**

Tahap ini merupakan lanjutan dari tahap sebelumnya dimana struktur data *skeleton* dan *motion* telah berhasil dibuat. Langkah selanjutnya adalah melakukan konversi lokasi *joint* dari posisi *lokal space* dalam bentuk matriks ke posisi *world space* dalam bentuk vektor. Perhitungan pusat massa setiap *segment* dapat dilakukan berdasarkan pembahasan pada tabel 2.1 dan tabel 2.2 menggunakan data yang didapat dari lokasi *joint* pada *world space*. Letak pusat massa tubuh secara keseluruhan dapat dihitung dengan menggunakan data lokasi pusat massa *segment*. Langkah-langkah diatas ditampilkan pada gambar 3.8.



***Gambar 3.8 Langkah-Langkah Penghitungan Pusat Massa***

Data yang tersimpan didalam *motion* merupakan informasi setiap *joint* pada *local space*. Suatu posisi *local space* berarti posisi tersebut relatif terhadap induknya, sedangkan posisi *world space* menandakan bahwa posisi tersebut relatif terhadap titik O (0.0, 0.0, 0.0). Konversi dari *local space* ke *world space* dapat dilakukan dengan melakukan operasi perkalian matriks dari induk teratas hingga ke *joint* yang ingin dicari secara rekursif. Posisi *world space* dari *head joint* dapat dicari dengan menggunakan:

MheadWorld = MhipLocal.MspineLocal.Mspine1Local.MneckLocal.MheadLocal (3.1)

yang berarti untuk mendapatkan posisi head pada *world space* dapat dilakukan dengan melakukan operasi perkalian matriks dari *hip* sampai ke *head*.

Data hasil penghitungan tersebut kemudian disimpan didalam dua buah bentuk *array*. *Array* pertama bernama bvhVertices menampung data posisi setiap joint, sedangkan *array* kedua bernama bvhIndices berisi indeks nomor setiap joint dengan ketentuan yang seperti pada tabel 3.3.

**Tabel 3.3 Isi BVHVertices dan BVHIndices**

|  |  |  |
| --- | --- | --- |
| BVHVertices | BVHIndices | Arti Indeks |
| (x, y, z) | 0 | Hips |
| (x, y, z) | 1 | Spine |
| (x, y, z) | 2 | Spine1 |
| (x, y, z) | 3 | Neck |
| (x, y, z) | 4 | Head |
| (x, y, z) | 5 | HeadEndSite |
| (x, y, z) | 6 | LeftShoulder |
| (x, y, z) | 7 | LeftArm |
| (x, y, z) | 8 | LeftForeArm |
| (x, y, z) | 9 | LeftHand |
| (x, y, z) | 10 | LeftHandEndsite |
| (x, y, z) | 11 | RightShoulder |
| (x, y, z) | 12 | RightArm |
| (x, y, z) | 13 | RightForeArm |
| (x, y, z) | 14 | RightHand |
| (x, y, z) | 15 | RightHandEndSite |
| (x, y, z) | 16 | LeftUpLeg |
| (x, y, z) | 17 | LeftLeg |
| (x, y, z) | 18 | LeftFoot |
| (x, y, z) | 19 | LeftTowBase |
| (x, y, z) | 20 | LeftTowBaseEndSite |
| (x, y, z) | 21 | RightUpLeft |
| (x, y, z) | 22 | RightLeg |
| (x, y, z) | 23 | RightFoot |
| (x, y, z) | 24 | RightToeBase |
| (x, y, z) | 25 | RightToeBaseEndSite |

**3.4.4 Rendering Menggunakan Modern OpenGL**

**3.4.5 Plotting Histogram**

**3.5 Tahap Uji Coba**

Tahap uji coba terdiri dari dua aktivitas, yaitu pengujian aplikasi dan pengujian aplikasi terhadap data lain. Tahap uji coba akan menentukan apakah aplikasi yang dibuat dapat menjadi solusi dari permasalah awal.