IMPLEMENTASI REAL TIME SKIN DEFORMATION

TUGAS AKHIR

Disusun Oleh :

BAWENANG RUKMOKO PARDIAN PUTRA

5100 100 043

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

2006

IMPLEMENTASI REAL TIME SKIN DEFORMATION

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan

Untuk Memperoleh Gelar Sarjana Komputer

pada

Jurusan Teknik Informatika

Fakultas Teknologi Informatika

Institut Teknologi Sepuluh Nopember

Surabaya

Mengetahui / Menyetujui

Dosen Pembimbing I	Dosen Pembimbing II
Ir. Esther Hanaya, M.Sc.	Rully A. Hendrawan, S. Kom.
NIP: 130 816 212	

SURABAYA

JANUARI, 2006

ABSTRAK

Dewasa ini dunia grafika komputer mengalami perkembangan yang pesat, terutama dalam hal animasi tiga dimensi. Skeletal animation adalah satu bentuk animasi tiga dimensi dengan menggunakan suatu model berkerangka dapat berjalan secara real-time. Animasi dijalankan berdasarkan gerakan kerangka, dan kulit akan mengikuti gerakan kerangka tersebut.

Teknik skeletal animation ini dikembangkan karena tidak terlalu memerlukan hardware dan resource yang berlebihan. Pergerakan kerangka akan menyebabkan kulit mengalami perubahan. Hal ini dinamakan skin deformation atau skinning. Pada basic skeletal animation, skin deformation yang dilakukan mengakibatkan kulit menjadi tidak halus . Hal ini sudah dapat diatasi dengan ditemukannya metode skin deformation yang bernama vertex blending. Tetapi, vertex blending juga mempunyai kelemahan. Dalam beberapa postur skeleton, vertex blending akan dapat mengakibatkan sebuah efek "melintir" pada skin, terutama pada postur dimana terjadi rotasi 180 derajat. Terdapat beberapa metode skin deformation lain yang dikembangkan untuk mengatasi efek melintir ini. Salah satunya yang akan diketengahkan disini adalah metode bones blending.

Tugas akhir ini mengetengahkan implementasi beberapa metode skin deformation yaitu basic skeletal animation, vertex blending, dan bones blending. untuk melihat kebenaran dari masing-masing metode. Uji coba dilakukan untuk mengetahui kelemahan dan keunggulan masing-masing metode.

Kata Kunci : *Skeletal animation, skin deformation, skinning.*

KATA PENGANTAR

Dengan memanjatkan puji syukur kehadirat Tuhan yang Maha Esa, karena atas berkat dan rahmatNyalah penulis dapat menyelesaikan Tugas Akhir ini guna memenuhi sebagian dari syarat-syarat untuk memperoleh gelar Sarjana Komputer jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya. Tugas Akhir ini mengetengahkan implementasi beberapa metode *skin deformation* secara *real time*, yaitu *basic skeletal animation, vertex blending,* dan *bones blending*.

Pada kesempatan ini, penulis ingin menyampaikan terima kasih yang sebesar-besarnya kepada:

1. Tuhan yang Maha Esa atas rahmat dan karunianya sehingga Tugas Akhir ini dapat terselesaikan.
2. Keluarga tercinta. Ayah, Ibu, dan adik-adik penulis yang telah memberikan dukungan secara moral. Terutama Ibu atas masukan dan "ancaman" yang membuat penulis terpacu dalam mengerjakan Tugas Akhir ini.
3. Ibu Ir. Esther Hanaya, M.Sc. selaku dosen pembimbing pertama yang telah memberikan bimbingan, dan bantuan selama penulis menyelesaikan Tugas Akhir sehingga Tugas Akhir ini dapat terselesaikan dengan baik.
4. Bapak Rully A. Hendrawan, S. Kom. selaku dosen pembimbing kedua yang telah memperkenalkan sebuah *API* bernama *OGRE 3D*

serta memberikan segala bantuan dan bimbingan untuk membantu penulis mengerjakan Tugas Akhir ini.

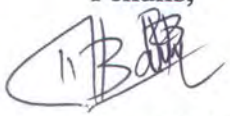
5. Ladislav Kavan atas bantuan dan kesabarannya ketika penulis menanyakan perihal *bones blending* melaui e-mail.
6. Sinbad, discipline, genva, xavier, dan lain-lain pada website OGRE 3D dan OGRE 3D forum atas segala bantuannya terutama dalam membantu penulis lebih dapat menggunakan *OGRE 3D*.
7. Bapak Yudhi Purwananto, S.Kom, M.Kom selaku ketua jurusan Teknik Informatika, FTIF – ITS.
8. Bapak Darlis Heru Mukti, S. Kom. selaku dosen wali yang telah memberikan arahan dan bantuan selama penulis menyelesaikan kuliah di jurusan Teknik Informatika, FTIF – ITS.
9. Bapak Ir. Khakim Ghozali selaku dosen wali pada awal-awal tahun penulis menjadi mahasiswa yang telah memberikan arahan dan bantuan selama penulis menyelesaikan kuliah di jurusan Teknik Informatika, FTIF – ITS, pada dua tahun awal masa perkuliahan penulis.
10. Seluruh Bapak dan Ibu dosen yang telah memberikan kuliah dan mendidik Penulis selama menuntut ilmu di ITS.
11. Teman-teman di Teknik Informatika, terutama teman-teman C10, yang namanya tidak bisa penulis sebutkan satu-persatu atas dukungan dan kerjasamanya selama ini.

12. Anak-anak semua lab di Teknik Informatika, terutama para administrator, yang memberikan dukungan dan semangat serta bantuan dalam pengerjaan Tugas Akhir ini.
13. Semua pihak yang tidak dapat penulis sebutkan satu-persatu yang telah membantu dan memberikan dukungan baik secara moral dan spiritual kepada penulis selama pengerjaan Tugas Akhir ini.

Penulis menyadari bahwa penyusunan Tugas Akhir ini tidak luput dari kesalahan dan kekurangan, sehingga penulis mengharapkan segala saran dan kritik yang sifatnya membangun. Akhir kata, semoga Tugas Akhir ini dapat memberikan manfaat dan memperluas wacana ilmu pengetahuan serta wawasan dalam bidang Teknik Informatika.

Surabaya, 18 Januari 2006

Penulis,



Bawenang R. P. P.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

BAB I
PENDAHULUAN

1.1. Latar Belakang

Dewasa ini dunia grafis komputer (*Computer Graphics* / CG) mengalami perkembangan yang sangat drastis. Terutama dengan ditemukannya teknologi 3 dimensi (dalam hal ini akan disingkat menjadi *3D*) yang sangat membantu dalam hal simulasi pada sebuah dunia *virtual*.

Salah satu bentuk perkembangan ini adalah dalam hal animasi sebuah model yang berkerangka. Animasi dengan cara melakukan transformasi pervertex pada sebuah model berkerangka kurang efektif apabila dilakukan secara *real time*. Tidak seperti sebuah animasi *pre-rendered*, animasi secara *real time* mempunyai batasan-batasan karena terbatasnya *resource* seperti CPU dan memori. Untuk mengatasi batasan-batasan ini, dalam menganimasikan sebuah model berkerangka dikembangkan suatu teknik animasi yang bernama *skeletal animation* untuk menganimasikan sebuah model *3D* berkerangka (terutama *humanoid* / sesuatu yang bagian tubuhnya menyerupai manusia[1]).

Teknik ini banyak digunakan pada permainan komputer *3D* yang membutuhkan banyak model berkerangka terutama permainan dengan *genre first-person*. Dalam *genre* ini, pemain akan mengeksplorasi sebuah dunia *3D* dalam pandangan orang pertama (atau lebih mudahnya pemain akan bermain seolah-olah dirinya berada dalam dunia *3D* tersebut dan pemain dapat merasakan bahwa

[1] http://en.wikipedia.org/wiki/Humanoid

objek-objek yang terlihat oleh karakter yang dimainkan dalam permainan tersebut adalah objek-objek yang terlihat oleh dirinya). Karena permainan berjalan secara *real time*, maka diperlukan kecepatan dalam melakukan *render* tampilan. Sehingga teknik animasi *pre-rendered* tidak cocok untuk permainan tipe ini.

Maka dalam menganimasikan sebuah model berkerangka secara *real time* seperti dalam sebuah permainan *3D*, sering digunakan teknik *skeletal animation.*

Teknik *skeletal animation* ini dikembangkan karena tidak terlalu memerlukan *hardware* dan *resource* yang berlebihan. Selain itu, dengan menggunakan sistem kerangka dari menyimpan posisi setiap verteks per-*frame*, bukan saja dapat menghemat kapasitas penyimpanan, tetapi juga dapat membuat gerakan baru secara instan. Kekurangan teknik ini adalah: pada *basic skeletal animation*, akan ditemukan kulit yang tidak halus pada beberapa postur *skeleton.* Hal ini sudah dapat diatasi dengan ditemukannya metode *skin deformation* yang bernama *vertex blending*. Tetapi, *vertex blending* juga mempunyai kelemahan. Dalam beberapa postur skeleton, *vertex blending* akan dapat mengakibatkan sebuah efek "melintir" pada *skin,* terutama pada postur dimana terjadi rotasi 180 derajat. Hal ini sudah dapat ditangani oleh metode *improved vertex blending* dan *bones blending*.

Tugas Akhir ini akan mengetengahkan metode-metode *skin deformation*, terutama *bones blending*. *Bones blending* diketengahkan pertama kali oleh Ladislav Kavan pada salah satu publikasinya yang berjudul "*Real Time Skin Deformation with Bones Blending*" [01]. Latar belakang dipilihnya metode *skin deformation*, terutama *bones blending*, ini adalah karena metode ini akan bisa

menampilkan model berkerangka yang lebih mirip aslinya dan dengan kebutuhan *hardware* dan *resource* yang relatif rendah.

1.2. Tujuan

Pembuatan Tugas Akhir ini bertujuan untuk mengimplementasikan metode-metode skin deformation yaitu: *basic skeletal animation* (*single weighted blending*), *vertex blending* (*multi weighted blending*), dan *bones blending* serta menganalisis kebenaran tampilan, kelemahan dan keunggulan masing-masing.

1.3. Permasalahan

Permasalahan yang dihadapi dalam mengimplementasikan sistem ini adalah antara lain :

1. Bagaimana merancang dan/atau mendapatkan sebuah model *3D* berkerangka dan memuatnya pada aplikasi.
2. Bagaimana mengimplementasikan *basic skeletal animation* pada sebuah model *3D*.
3. Bagaimana mengimplementasikan metode *vertex blending* pada model *3D*
4. Bagaimana mengimplementasikan metode *bones blending* pada model *3D*.

1.4. Batasan Masalah

Sejumlah batasan masalah yang akan dibahas dalam Tugas Akhir ini adalah sebagai berikut :

1. Metode *skin deformation* yang akan diimplementasikan adalah *Basic Skeletal Animation, Vertex Blending,* dan *Bones Blending.*

2. Metode/algoritma pendukung *skin deformation* memakai referensi *open source*.
3. Implementasi dalam pemecahan masalah memanfaatkan atau mengintegrasikan pustaka *open source*

1.5. Metodologi

Metodologi yang digunakan untuk menyelesaikan tugas akhir ini adalah sebagai berikut:

a. Studi Kepustakaan

Pada tahap ini dilakukan pemahaman kepustakaan yang berhubungan dengan pembuatan perangkat lunak secara umum, seperti teori-teori Grafika, *Skeletal Animation*, *Vertex Blending*, *Bones Blending*, *Spherical Linear Interpolation (SLERP)*, dan lain-lain.

b. Desain sistem

Tahap ini meliputi perancangan sistem dengan menggunakan studi literatur dan mempelajari konsep teknologi dari software yang ada. Dalam tahap ini, akan dilakukan desain antar muka, pencarian model *3D*, desain proses, dan desain aplikasi secara keseluruhan.

c. Implementasi

Pada tahap ini dilakukan implementasi rancangan sistem yang telah dibuat. Tahapan ini merealisasikan apa yang terdapat pada tahapan desain menjadi sebuah aplikasi yang sesuai dengan apa yang direncanakan.

d. Uji Coba dan Evaluasi

Pada tahap ini, dilakukan uji coba terhadap ketahanan aplikasi dengan beberapa data untuk kemudian dilakukan perbaikan apabila terdapat kesalahan sehingga dapat dilakukan evaluasi terhadap hasil uji coba tersebut.

e. Penyusunan laporan Tugas Akhir

Tahap ini dilakukan untuk membuat laporan dari semua dasar teori dan metode yang digunakan serta hasil-hasil yang diperoleh selama pengerjaan tugas akhir.

1.6. Sistematika Pembahasan

Pembahasan dalam Tugas Akhir ini dibagi menjadi beberapa bab sebagai berikut :

- Bab I, Pendahuluan, berisi latar belakang, permasalahan, tujuan, batasan masalah, metodologi dan sistematika pembahasan.
- Bab II, Pencarian informasi pada dokumen terstruktur, akan dibahas dasar ilmu yang berhubungan dengan pencarian informasi pada dokumen terstruktur. Seperti berbagai hal mengenai skeletal animation, quaternion, Spherical Linear Interpolation (SLERP), vertex blending, bones blending, dan teori-teori penunjang lainnya.
- Bab III, Perancangan Perangkat Lunak. Pada bab ini akan dibahas rancangan proses dan rancangan antarmuka.
- Bab IV, Implementasi Perangkat Lunak. Akan dilakukan pembuatan aplikasi yang dibangun dengan komponen-komponen yang telah ada yang sesuai dengan permasalahan dan batasannya yang telah dijabarkan pada bab pertama beserta implementasinya.

- Bab V, Uji coba dan analisis hasil, akan dilakukan uji coba berdasarkan parameter-parameter yang ditetapkan, dan kemudian dilakukan analisis terhadap hasil uji coba tersebut.
- Bab VI, Penutup, berisi kesimpulan yang dapat diambil dari Tugas Akhir ini beserta saran untuk pengembangan selanjutnya.

BAB II
DASAR TEORI

BAB II
DASAR TEORI

Pada bab ini akan dibahas tentang berbagai dasar teori yang dipakai oleh penulis dalam pembuatan Tugas Akhir ini. Diantaranya adalah mengenai teknik *skeletal animation*, representasi rotasi, *Spherical Linear intERPolation (SLERP)*, dan metode *skin deformation* seperti *vertex blending*, dan *bones blending*.

2.1. Skeletal Animation

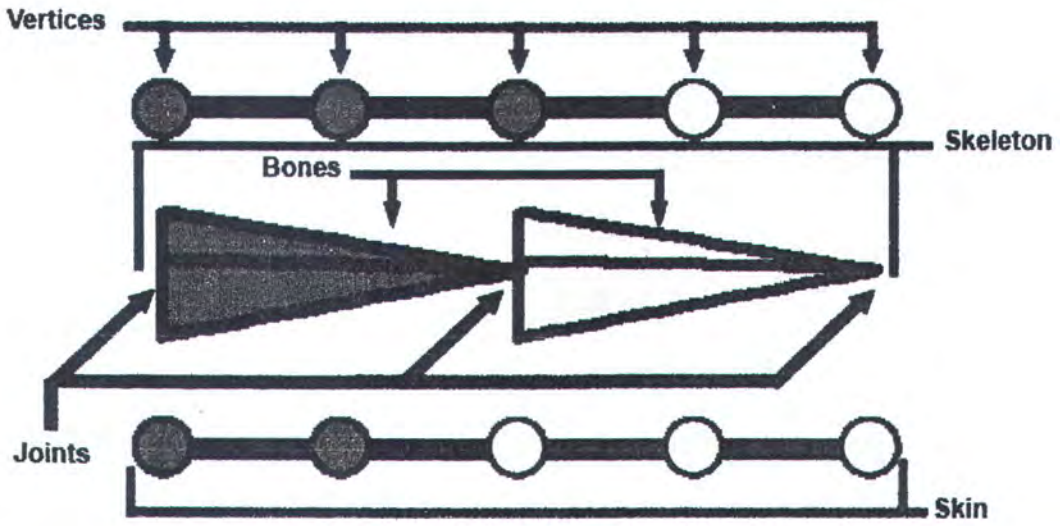
Seperti yang telah dibahas sebelumnya, metode *skeletal animation* ini dikembangkan untuk permainan komputer *3D*. Jadi, referensi-referensi yang ada tentang *skeletal animation* ini lebih banyak terdapat pada artikel-artikel mengenai permainan komputer yaitu: [03], [04], dan [05] serta [06]. *Skeletal animation* pada dasarnya adalah membagi sebuah model menjadi[2]:

- *Joints* (sendi) : node-node yang direpresentasikan oleh matriks homogen yang berfungsi untuk pergerakan rotasi dan translasi.
- *Bones* (tulang) : bagian-bagian yang menghubungkan dua buah *joints* (sendi). Sebuah tulang biasanya didentifikasikan dengan sendi awal dari tulang tersebut. Pada Tugas Akhir ini, tulang merepresentasikan sendi *parent* dari tulang tersebut. Tulang dan sendi akan dipakai saling menggantikan.

[2] Kavan, L. *Real Time Skin Deformation with Bones Blending.* In WSCG Short Proceeding, 2003, hal.1.

- *Skeleton* (kerangka) : keseluruhan badan model yang berbentuk pohon (*tree*). Node-node dari *tree* tersebut adalah *joints* (sendi) dan edge-edge dari *tree* adalah *bones* (tulang). Akar (*root*) dari *tree* disebut juga *root joint*.
- *Skin* (kulit) : *mesh* yang terdiri dari beberapa *polygon* (biasanya berbentuk segitiga). Setiap verteks dari setiap *polygon* hanya dapat dipasangkan hanya pada satu tulang. Transformasi verteks akan mengikuti transformasi tulang pasangannya.

Penggambaran lebih jelas dari model ini dapat dilihat pada gambar 1 berikut:



Gambar 1. Pembagian model berkerangka

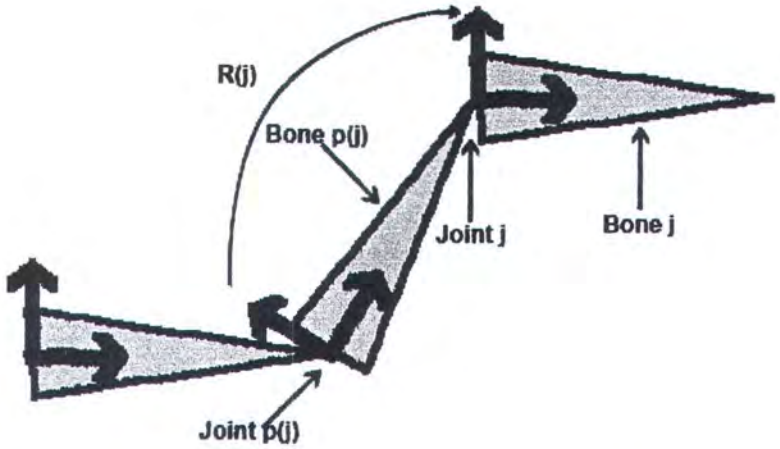
Pada *skeletal animation*, sebuah model direpresentasikan oleh sebuah kerangka dimana sekumpulan kulit akan dipasangkan pada kerangka tersebut. Kerangka tersebut terdiri dari beberapa tulang dan sendi. Antara dua tulang, akan dihubungkan oleh satu sendi. Sendi adalah poros tempat proses transformasi

tulang berlangsung (biasanya berupa rotasi dan translasi). Setiap sendi dan tulang merupakan *child* (anak) dari sendi dan tulang di atasnya (kecuali *root joint* yang merupakan permulaan dari hierarki kerangka). *Root joint* adalah sendi awal yang merupakan pusat pergerakan seluruh kerangka (berdasarkan konvensi umum, *root joint* biasanya ditempatkan di tengah-tengah pinggul kanan dan kiri/ pertemuan tulang punggung dan pinggul). Setelah sendi-sendi dan tulang-tulang tergabung dan membentuk suatu kerangka, maka selanjutnya akan dipasang kulit pada kerangka. Kulit terdiri dari sekumpulan *mesh* berbentuk segitiga yang setiap verteksnya hanya dapat dipasangkan pada satu tulang. Apabila terjadi transformasi pada tulang, maka setiap *vertex* dari kulit akan bertransformasi juga mengikuti transformasi pada tulang yang telah dipasangkan pada *vertex* tersebut. Pada aplikasinya di *computer graphics*, sebuah tulang akan direpresentasikan oleh sendi *parent* dari tulang tersebut (jadi pada prakteknya, nilai dan variabel dari sebuah tulang adalah berupa nilai dan variabel dari sendi *parent*).

Setiap kerangka mempunyai sebuah posisi awal yang disebut posisi referensi. Umumnya posisi referensi ini berbentuk posisi penyaliban (*crucifixion*).

Setiap sendi dapat mempunyai sebuah sistem koordinat lokal. Untuk menentukan posisi dan orientasi sendi ini, diperlukan suatu matriks untuk menghitung rotasi sendi j dari koordinat referensi yang dinotasikan dengan $A(j)$. Posisi dan orientasi sendi j sangat ditentukan oleh posisi dan orientasi sendi *parent* di atasnya yang dinotasikan dengan $p(j)$ dan sendi $p(j)$ juga ditentukan oleh sendi di atasnya lagi. Hal ini berlangsung secara rekursif sampai ke *root joint* atau $j=0$.

Untuk lebih jelasnya, bisa dilihat pada gambar berikut ini:



Gambar 2. Contoh transformasi posisi referensi sendi *j*

Pada gambar di atas dimisalkan sebuah sendi *j* terdapat pada sebuah model kerangka. Transformasi terjadi pada sendi *j* dan *p(j)* yang merupakan sendi *parent* dari sendi *j*. Transformasi total *A(j)* relatif terhadap koordinat dunia adalah:

$$A(j) = R(0)...R(p(j))R(j) \quad (1)$$

dimana:

A(j) = Matriks transformasi total untuk menempatkan sendi *j* dari koordinat origin dunia secara hierarki

R(0) = Matriks transformasi *root joint* terhadap koordinat dunia

R(j) = Matriks transformasi sendi *j* terhadap sistem koordinat lokal *p(j)* pada posisi referensi

Keseluruhan transformasi pada sendi *j* yang terjadi secara garis besar dapat dijelaskan sebagai berikut:

1. Aplikasikan transformasi pada *root joint*.
2. Aplikasikan transformasi pada sendi *child* di bawahnya.

3. Ulang langkah 2 sampai pada sendi *j*.

Untuk menganimasikan kerangka, diperlukan sebuah matriks transformasi yang berisi transformasi yang terjadi pada sendi *j* sesungguhnya. Matriks transformasi ini dinotasikan dengan *T(j)*. Transformasi pada sebuah sendi tidak boleh berupa selain rotasi. Transformasi yang berupa translasi akan menyebabkan suatu keadaan tidak normal pada kerangka karena akan menyebabkan panjang tulang menjadi berubah. Translasi hanya dapat diterapkan pada *root joint*. Transformasi pada *root joint* berarti transformasi pada seluruh kerangka. Sedangkan transformasi lain seperti *scaling* dan *shear* tidak akan digunakan karena akan menyebabkan perubahan bentuk dari model berkerangka. Matriks transformasi final dari suatu posisi kerangka yang teranimasi didefinisikan sebuah transformasi final *F(j)*:

$$F(j) = F(p(j))R(j)T(j)$$
$$F(0) = R(0)T(0) \qquad (2)$$

dimana:

F(j) = Matriks transformasi final sendi *j* terhadap sistem koordinat dunia dari suatu posisi kerangka teranimasi.

R(j) = Matriks transformasi sendi *j* terhadap sistem koordinat lokal *p(j)* pada posisi referensi

T(j) = Matriks rotasi yang terjadi pada sendi *j* sesungguhnya.

Dengan *skeletal animation*, dalam menentukan posisi verteks setelah terjadi suatu animasi tidak memerlukan transformasi per verteks melainkan hanya mentransformasikan kerangka suatu model dan verteks dari kulit model tersebut akan mengikuti transformasi dari kerangka. Untuk menentukan penyebaran

transformasi dari kerangka ke kulit, hubungan antara kerangka dan kulit harus didefinisikan terlebih dahulu. Pendekatan yang paling dasar (*basic skeletal animation*) adalah, satu verteks dari kulit hanya dapat dipasangkan pada satu tulang dan hanya tulang tersebut yang dapat mempengaruhi posisi verteks. Verteks akan bertransformasi berdasarkan transformasi tulang dimana verteks tersebut ditempatkan. Jika suatu verteks $v \in R^3$ dipasangkan pada sebuah tulang dengan sendi j, maka posisi verteks v' yang melakukan transformasi adalah:

$$v' = F(j)A(j)^{-1}v \tag{3}$$

Penjelasan formula di atas adalah sebagai berikut: matriks $A(j)^{-1}$ mentransformasikan verteks v dari koordinat referensi (dunia) ke koordinat lokal sendi j. Ini disebabkan oleh $T(j)$ dalam matriks $F(j)$ bekerja pada sendi j. Perkalian dengan $F(j)$ dapat diartikan sebagai berikut: pertama terapkan rotasi sendi $T(j)$ lalu lalu kembalikan verteks yang terotasi dari sistem koordinat lokal sendi j ke sistem koordinat referensi (dunia) dengan $R(j)$. Jika semua $T(j)$ dalam $F(j)$ merupakan matriks identitas, maka tidak terjadi transformasi dan $v=v'$. Matriks $A(j)^{-1}$ tidak akan berubah (karena posisi referensi tidak akan berubah), karena itu dapat dihitung sebelumnya dan nilainya disimpan sebagai variabel untuk menghemat proses.

Pada aplikasi *skeletal animation* yang sebenarnya dalam sebuah perangkat lunak, sebuah model kerangka dapat direpresentasikan oleh sekumpulan sendi-sendi. Berdasarkan konvensi, pangkal hierarki(*root joint*) berada pada bagian perut bawah atau perpotongan antara tulang kaki atas dan punggung bawah. Sendi itu sendiri direpresentasikan oleh sebuah vektor yang terdiri dari sebuah matriks

4X4. Matriks 4X4 tersebut menyimpan bagian rotasi dan translasi dari sendi. Bagian rotasi menentukan orientasi dari sendi tersebut relatif terhadap sendi *parent*. Dan bagian translasi menyimpan posisi sendi tersebut relatif terhadap sendi *parent*. Matriks 4X4 tersebut dinamakan matriks relatif (matriks *R(j)*). Dari matriks tersebut, dapat dihitung sebuah matriks absolut (matriks *A(j)*) secara hierarkikal. Matriks absolut adalah matriks untuk mendapatkan posisi sendi tersebut dari koordinat origin (0,0,0) dunia. Matriks relatif adalah matriks untuk mendapatkan posisi sendi tersebut dari sendi *parent*.

Sedangkan untuk merepresentasikan tulang juga direpresentasikan oleh sendi, yaitu sendi *parent* dari tulang tersebut. Untuk mencari panjang tulang, dapat dilakukan dengan menghitung jarak antara sendi *parent* dengan sendi *child* dari tulang tersebut. Karena sendi *child* tulang adalah sendi *parent* dari tulang di bawahnya (tulang *child*), maka untuk mengetahui panjang tulang dapat dilakukan dengan mencari jarak antara tulang terhadap tulang di bawahnya. Operasi yang dapat dilakukan pada tulang adalah operasi rotasi saja. Apabila terjadi operasi translasi, maka jarak antara tulang dan tulang *parent* akan berubah dan akan mengakibatkan perubahan bentuk model. Karena itu transformasi yang diperbolehkan pada sendi selain *root joint* hanya berupa rotasi saja.

Gambar 3. Contoh representasi tulang

Pada gambar contoh representasi tulang di atas, tulang *bone1* direpresentasikan oleh sendi *joint1*. Bagian rotasi dari *joint1* menyatakan orientasi dari *bone1*. Sedangkan bagian translasi dari *joint1* menyatakan posisi *bone1* terhadap tulang sebelumnya. Selain itu, jarak (magnitude) bagian translasi *joint1* menyatakan panjang tulang *parent* dari *bone1*. Panjang tulang *bone1* itu sendiri dinyatakan oleh jarak dari bagian translasi *joint2*. Demikian juga *bone2* direpresentasikan oleh *joint2* dan panjang *bone2* adalah jarak dari bagian translasi *joint3*.

2.2. Representasi Rotasi

Transformasi yang terjadi pada *skeletal animation* hanya berupa translasi (pada *root joint*) dan rotasi. Karena sebagian besar operasi transformasi pada *skeletal animation* berupa rotasi, maka pada subbab ini akan dijelaskan lebih lanjut tentang representasi rotasi. Rotasi yang akan dijelaskan di dalam Tugas Akhir ini adalah rotasi yang arah perputarannya berlawanan arah jarum jam. Rotasi dapat direpresentasikan dengan beberapa cara antara lain representasi matriks, representasi sudut *Euler*, representasi sudut-sumbu (*Angle-Axis Representation*), dan *quaternion*. Untuk penjelasan lebih jauh dari representasi rotasi, dapat dilihat pada [16]. Representasi rotasi yang akan dipakai pada Tugas Akhir ini adalah bentuk *quaternion*.

2.2.1. Quaternion

Sebuah metode yang lebih efisien dalam mencari sebuah rotasi berdasarkan sebuah sumbu rotasi tertentu adalah dengan menggunakan

representasi *quaternion*. *Quaternion* juga mempunyai banyak fungsi dalam grafika komputer seperti membuat fraktal tiga dimensi, dan perhitungan rotasi tiga dimensi. *Quaternion* membutuhkan tempat penyimpanan yang lebih kecil daripada sebuah matriks 4X4 dan lebih memudahkan pembuatan prosedur rangkaian transformasi. Umumnya, *quaternion* banyak digunakan pada animasi tiga dimensi yang membutuhkan banyak rangkaian gerakan dan interpolasi antara dua buah posisi *key frame*.

Konsep *quaternion* itu sendiri berasal dari generalisasi bilangan kompleks yang pertama kali ditemukan oleh Mark Hamilton.

Quaternion dapat dinyatakan dengan sebuah bilangan dengan sebuah bagian *Real* (riil) dan tiga buah bagian imajiner yang dapat dituliskan menjadi:

$$q = w + xi + yj + zk$$
$$q = (s, v) \quad (4)$$

dimana:

s = w= Bagian skalar / riil dari *quaternion*

v = $xi+yj+zk$= Bagian vektor / imajiner dari *quaternion*

Sebuah *quaternion* dapat menentukan sebuah titik pada sebuah ruang R^4 (empat dimensi / *4D*), atau jika s=0, maka merupakan sebuah titik pada ruang tiga dimensi (*3D*). *Quaternion* dapat digunakan untuk merepresentasikan sebuah vektor beserta rotasinya. Bilangan-bilangan imajiner *i,j,* dan *k* merupakan sebuah *quaternion* unit yang ekuivalen dengan vektor unit pada sistem vektor. Tetapi, bilangan-bilangan tersebut memenuhi peraturan kombinasi yang berbeda:

$$i^2 = j^2 = k^2 = ijk = -1, ij = -ji = k \quad (5)$$

Berdasarkan peraturan [42] di atas, maka dapat ditentukan:

$$jk = -kj = i, ki = -ik = j \tag{6}$$

Operasi penjumlahan *quaternion* adalah sebagai berikut:

$$q + q' = (s + s', v + v') = (s + s') + (x + x')i + (y + y')j + (z + z')k \tag{7}$$

Sedangkan operasi perkalian *quaternion* adalah sebagai berikut:

$$q.q' = (s.s' - v \bullet v', v \times v' + s.v' + s'.v) \tag{8}$$

Sebagai pengembangan dari operasi bilangan kompleks, maka untuk menghitung jarak dari *quaternion* adalah:

$$|q| = \sqrt{s^2 + v \bullet v} \tag{9}$$

Dan invers dari quaternion adalah:

$$q^{-1} = \frac{1}{|q|^2}(s, -v) \tag{10}$$

Sehingga:

$$q.q^{-1} = q^{-1}.q = (1,0) \tag{11}$$

Sebuah rotasi terhadap sembarang sumbu rotasi yang melalui titik pangkal dapat dilakukan dengan membuat sebuah *quaternion* unit dengan bagian skalar dan vektor sebagai berikut:

$$s = \cos\frac{\theta}{2}, v = u.\sin\frac{\theta}{2} \tag{12}$$

dimana u adalah sebuah vektor unit yang berada pada sumbu rotasi yang diinginkan dan θ adalah sebuah sudut rotasi terhadap sumbu rotasi tersebut. Sebuah titik P dapat direpresentasikan dengan notasi *quaternion* sebagai berikut:

$$P = (0, p) \tag{13}$$

Dengan koordinat titik P tersebut pada ruang tiga dimensi adalah bagian vektor $p=(x,y,z)$. Rotasi dari titip P tersebut dapat dinyatakan dengan operasi *quaternion* sebagai berikut:

$$P' = qPq^{-1} \tag{14}$$

dimana $q^{-1} = (s,-v)$ adalah invers dari *quaternion* q dengan bagian skalar dan vektor seperti yang telah ditentukan. Transformasi ini menghasilkan sebuah *quaternion* baru dengan bagian skalar 0:

$$P' = (0, p') \tag{15}$$

Dan bagian vektor p' dihitung dari:

$$p' = s^2 p + v(p \bullet v) + 2s(v \times p) + v \times (v \times p) \tag{16}$$

Parameter s dan v mempunyai nilai rotasi seperti pada persamaan di atas. Transformasi pada persamaan (16) ekuivalen dengan sebuah rotasi terhadap suatu sumbu rotasi yang melewati pangkal koordinat. Rotasi ini sama dengan serangkaian transformasi yang merotasikan sumbu rotasi dengan sumbu z, melakukan rotasi yang diinginkan terhadap sumbu z, lalu mengembalikan sumbu rotasi pada orientasinya semula.

Dengan menggunakan representasi *quaternion*, perhitungan rotasi dapat dilakukan hanya dengan mendefinisikan 4 parameter yaitu $a,b,c,$dan sudut θ. Jika dibandingkan dengan representasi sudut *Euler*, yang memakai 9 parameter untuk setiap nilai dalam matriks rotasinya, tentu saja dapat dilihat bahwa memakai *quaternion* lebih efisien dalam hal pemakaian memori. Hal ini juga yang menyebabkan representasi *quaternion* lebih banyak dipakai dalam menyatakan rotasi.

Dalam implementasinya, *quaternion* sering dikonversikan dari/ke representasi matriks rotasi 3x3. Konversi *quaternion* ini dapat dilihat pada [15]. Matriks rotasi ini adalah tiga baris dan tiga kolom pertama dari matriks transformasi 4x4 yang merupakan standar representasi transformasi dalam suatu ruang yang homogen.

Pada subbab-subbab berikut akan dijelaskan metode konversi *quaternion* dari / ke matriks rotasi 3X3 menurut [15].

2.2.1.1. Konversi Quaternion ke Matriks Rotasi

Dengan menggunakan identitas $2\sin^2(\theta/2) = 1-\cos\theta$ dan $\sin\theta = 2\sin(\theta/2)\cos(\theta/2)$ didapatkan nilai-nilai $2wx = (\sin\theta)w_0$, $2wy = (\sin\theta)w_1$, $2wz = (\sin\theta)w_2$, $2x^2 = (1-\cos\theta){w_0}^2$, $2xy = (1-\cos\theta)w_0w_1$, $2xz = (1-\cos\theta)w_0w_2$, $2y^2 = (1-\cos\theta){w_1}^2$, $2yz = (1-\cos\theta)w_1w_2$, dan $2z^2 = (1-\cos\theta){w_2}^2$. Sisi sebelah kanan dari persamaan-persamaan tersebut adalah istilah-istilah dari persamaan $M_R = I + (\sin\theta)S + (1-\cos\theta)S^2$. Dengan mengganti istilah-istilah tersebut didapatkan:

$$M_R = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix} \quad (17)$$

Matriks 3X3 pada persamaan (17) di atas adalah bagian rotasi dari sebuah matriks transformasi 4X4 pada suatu ruang homogen. Yaitu nilai r_{00}, r_{01}, r_{02}, r_{10}, r_{11}, r_{12}, r_{20}, r_{21}, dan r_{22} dari sebuah matriks transformasi R. Untuk menkonversi

dari sebuah matris rotasi 3X3 ke matriks transformasi 4X4, dilakukan pemberian nilai $r_{30}= r_{31}= r_{32}= r_{03}= r_{13}= r_{23} = 0$ dan nilai $r_{33} = 1$.

Untuk menyelesaikan rotasi yang diinginkan, sumbu rotasi masih harus di translasikan ke dalam kedudukan semula:

$$R(\theta) = T^{-1}.M_R.T \tag{18}$$

dimana:

$R(\theta)$ =Matriks transformasi akhir

T = Matriks translasi dari bagian vektor *quaternion* relatif terhadap koordinasi origin dunia.

M_R = Matriks rotasi *quaternion.*

2.2.1.2. Konversi Matriks Rotasi ke Quaternion

Untuk dapat menggunakan *quaternion* dalam menentukan rotasi, pertama yang harus dilakukan adalah melakukan konversi dari *quaternion* menjadi matriks rotasi. Dalam melakukan konversi dari matriks rotasi ke *quaternion*, pertama yang harus dilakukan adalah mencari nilai *trace* dari matriks rotasi. *Trace* adalah nilai total penjumlahan dari nilai-nilai diagonal matriks rotasi (total penjumlahan nilai r_{00}, r_{11}, dan r_{22}) dimana:

$$\cos\theta = (Trace(R)-1)/2 \tag{19}$$

Sehingga:

$$\theta = \cos^{-1}((Trace(R)-1)/2) \in [0,\pi] \tag{20}$$

Juga dapat dilihat bahwa:

$$R - R^T = (2\sin\theta)S \tag{21}$$

dimana S adalah sebuah matriks simetri miring (*skew-symmetric matrix*).

Dengan menggunakan identitas $2\cos^2(\theta/2) = 1+\cos\theta$ maka didapatkan nilai $w^2 = \cos^2(\theta/2) = (Trace(R)+1)/4$ dan nilai $|w| = \sqrt{Trace(R)+1}/2$. Terdapat dua kemungkinan nilai *Trace(R)* yang akan didapatkan.

Kemungkinan pertama adalah jika nilai *Trace(R)*>0, maka nilai $|w|>1/2$. Jadi, tanpa mengakibatkan kehilangan generalitas, pilih w sebagai akar kuadrat positif, $w = \sqrt{Trace(R)+1}/2$. Dari identitas (21) didapatkan $(r_{12}-r_{21}, r_{20}-r_{02}, r_{01}-r_{10}) = 2\sin\theta(w_0, w_1, w_2)$. Identitas yang didapatkan dari subbab sebelumnya adalah $2wx = (\sin\theta)w_0$, $2wy = (\sin\theta)w_1$, dan $2wz = (\sin\theta)w_2$. Dengan menggabungkan persamaan-persamaan tersebut didapatkan $x = (r_{12}-r_{21})/(4w)$, $y = (r_{20}-r_{02})/(4w)$, dan $z = (r_{01}-r_{10})/(4w)$.

Kemungkinan kedua adalah jika *Trace(R)*$\leq$0, maka nilai $|w|\leq 1/2$. Pertama yang dilakukan adalah mencari nilai paling besar dari tiga buah nilai diagonal r_{00}, r_{11}, dan r_{22}. Apabila r_{00} adalah nilai diagonal paling besar, maka x mempunyai jarak lebih besar daripada y atau z. Dengan beberapa operasi aljabar didapatkan nilai $4x^2 = r_{00}-r_{11}-r_{22}+1$ sehingga didapatkan nilai $x = \sqrt{r_{00}-r_{11}-r_{22}+1}/2$. Secara langsung dapat ditentukan nilai $w = (r_{12}-r_{21})/(4x)$, $y = (r_{01}+r_{10})/(4x)$, dan $z = (r_{02}+r_{20})/(4x)$. Apabila r_{11} adalah nilai diagonal paling besar, maka y mempunyai jarak lebih besar daripada x atau z. Dengan beberapa operasi aljabar didapatkan nilai $4y^2 = r_{11}-r_{00}-r_{22}+1$ sehingga didapatkan nilai $y = \sqrt{r_{11}-r_{00}-r_{22}+1}/2$. Secara langsung dapat ditentukan nilai

$w = (r_{20} - r_{02})/(4y)$, $x = (r_{01} + r_{10})/(4y)$, dan $z = (r_{12} + r_{21})/(4y)$. Apabila r_{22} adalah nilai diagonal paling besar, maka z mempunyai jarak lebih besar daripada x atau y. Dengan beberapa operasi aljabar didapatkan nilai $4z^2 = r_{22} - r_{00} - r_{11} + 1$ sehingga didapatkan nilai $z = \sqrt{r_{22} - r_{00} - r_{11} + 1}/2$. Secara langsung dapat ditentukan nilai $w = (r_{01} - r_{10})/(4z)$, $x = (r_{02} + r_{20})/(4z)$, dan $y = (r_{12} + r_{21})/(4z)$.

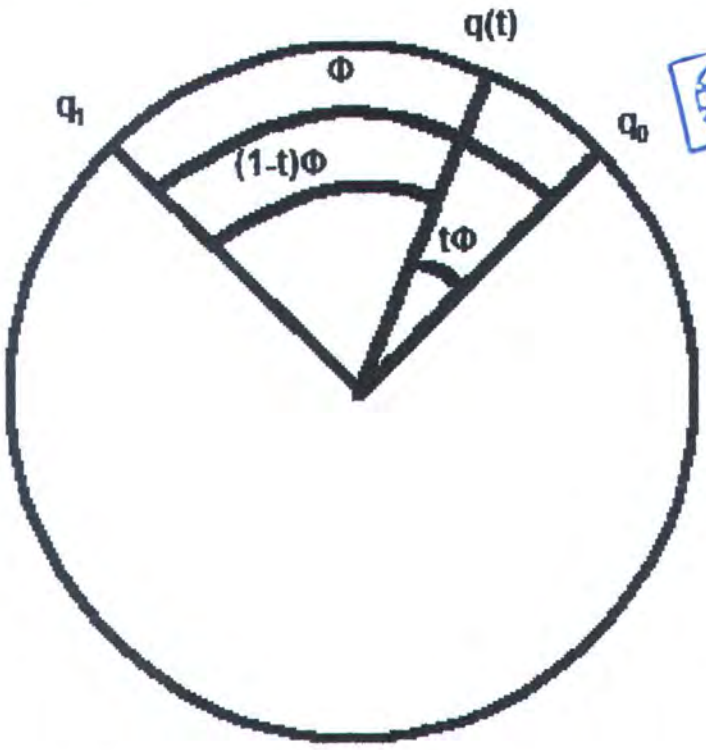
2.3. SLERP (Spherical Linear intERPolation)

Untuk menentukan sebuah posisi dalam suatu animasi, teknik yang umum digunakan adalah teknik interpolasi atau *key-framing*. Cara kerja teknik ini adalah menentukan beberapa buah posisi *key-frame* dari suatu animasi dan menentukan posisi-posisi lain di antara 2 *key-frame* dengan menghitung interpolasi posisi. Untuk interpolasi dalam *quaternion*, setiap kunci (*key*) direpresentasikan dalam sebuah matriks rotasi. Rangkaian matriks-matriks rotasi tersebut akan dikonversikan ke dalam rangkaian *quaternion-quaternion*. Interpolasi di antara *quaternion* kunci lalu dilakukan dan akan menghasilkan *quaternion-quaternion* diantaranya yang nantinya akan dikonversi ulang menjadi matriks-matriks rotasi. Matriks-matriks tersebut nantinya yang akan diimplementasikan ke dalam objek.

Sebuah permasalahan untuk menginterpolasikan antara dua buah matriks rotasi R_0 dan R_1 dengan pilihan-pilihan yang beragam yang memenuhi $t \in [0,1]$ adalah biasa dalam bidang grafika komputer. Penginterpolasi dinotasikan dengan $R(t)$, sebuah matriks rotasi itu sendiri, dan harus memenuhi $R(0)=R_0$ dan $R(1)=R_1$. Representasi *4-tuple* (w,x,y,z) dari sebuah matriks rotasi dan sebuah *hypersphere*

yang sesuai memperkenankan sebuah interpolasi yang sederhana tetapi efektif yaitu *Spherical Linear intERPolation* atau disingkat *slerp*.

Jika $q_i=(w_i,x_i,y_i,z_i)$ adalah sebuah representasi *4-tuple* untuk $R_i(i=0,1)$ dan $q(t)$ sebuah representasi *4-tuple* untuk $R(t)$, maka dapat ditentukan secara geometri bahwa $q(t)$ berada di antara busur *hypersphere* yang menghubungkan q_0 dan q_1. Selain itu, sudut antara $q(t)$ dan q_0 harus proporsional dengan sudut Φ yang berada antara q_0 dan q_1 dengan konstanta proporsi sebesar t. Gambar di bawah mengilustrasikan keadaan ini.



Gambar 4. Ilustrasi *SLERP* antara dua *quaternion*

Sudut Φ antara q_0 dan q_1 didapatkan secara tidak langsung dari perkalian skalar (*dot product*) $\cos(\Phi) = q_0 \bullet q_1$. Penginterpolasi diharuskan dalam bentuk $q(t) = c_0(t)q_0 + c_1(t)q_1$ dengan fungsi koefisien c_0 dan c_1 yang akan ditentukan

nanti. Selama t bervariasi antara 0 dan 1, nilai $q(t)$ juga bervariasi di antara busur melingkar antara q_0 dan q_1. Karena itu, sudut antara $q(t)$ dan q_0 adalah $t\Phi$, dan sudut antara $q(t)$ dan q_1 adalah $(1-t)\Phi$.

Dengan mengaplikasikan perkalian skalar antara $q(t)$ dan q_0 menghasilkan:

$$\cos(t\Phi) = c_0(t) + \cos(\Phi)c_1 t \tag{22}$$

Dan aplikasi perkalian skalar antara $q(t)$ dan q_1 menghasilkan:

$$\cos((1-t)\Phi) = \cos(\Phi)c_0(t) + c_1 t \tag{23}$$

Dari persamaan (22) dan (23), dapat dicari hasil fungsi c_0 dan c_1. Hasil dari fungsi c_0 adalah:

$$c_0(t) = \frac{\cos(t\Phi) - \cos(\Phi)\cos((1-t)\Phi)}{1-\cos^2(\Phi)} = \frac{\sin((1-t)\Phi)}{\sin(\Phi)} \tag{24}$$

Persamaan fungsi c_1 didapatkan dengan mengaplikasikan formula sudut ganda untuk sinus dan cosinus. Secara simetris, $c_1(t) = c_0(1-t)$. Dengan begitu, hasil dari fungsi c_1 adalah:

$$c_1(t) = \frac{\cos((1-t)\Phi) - \cos(\Phi)\cos(t\Phi)}{1-\cos^2(\Phi)} = \frac{\sin(t\Phi)}{\sin(\Phi)} \tag{25}$$

Persamaan *spherical linear interpolation* adalah:

$$SLERP\ (t; q_0, q_1) = \frac{\sin((1-t)\Phi)q_0 + \sin(t\Phi)q_1}{\sin(\Phi)} \tag{26}$$

, untuk setiap $0 \le t \le 1$.

Apabila 2 buah *quaternion* yang akan diinterpolasi adalah *quaternion* unit (misal *quaternion* unit P dan Q), maka *SLERP* dari 2 buah *quaternion* unit tersebut adalah:

$$SLERP(t; P, Q) = P(P^T Q)^t \tag{27}$$

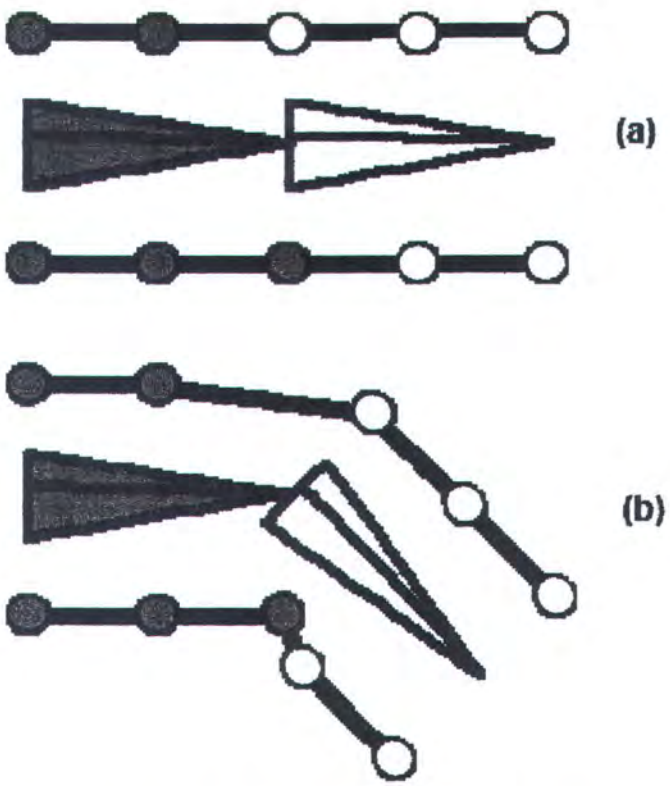
, untuk setiap $0 \leq t \leq 1$.

Jika nilai t=0, maka dapat dilihat bahwa $SLERP(0;P,Q)=P$. Dan jika nilai t=1, maka dapat dilihat bahwa $SLERP(1;P,Q)=Q$. Transpos dari matriks rotasi P (yang ortonormal dan mempunyai determinan positif) adalah sebuah matriks, dan perkalian matriks rotasi dengan matriks rotasi lain juga. Sehingga hasil dari *SLERP* juga akan berupa matriks rotasi.

Permasalahan muncul dalam menentukan notasi R^t dimana $R=P^TQ$, R adalah sebuah rotasi dan t adalah suatu bilangan *Real*. Matriks rotasi R dapat dikonversikan ke dalam bentuk sudut-sumbu. Apabila sebuah sumbu A dan sudut Φ dihasilkan dari konversi, maka R^t adalah sebuah rotasi pada sumbu rotasi A dengan sudut $t\Phi$.

2.4. Vertex Blending

Pada *basic skeletal animation*, ketika sebuah tulang melakukan pergerakan (misalnya rotasi), kulit di dekat sendi akan tampak tidak halus seperti yang dapat dilihat pada gambar berikut ini:

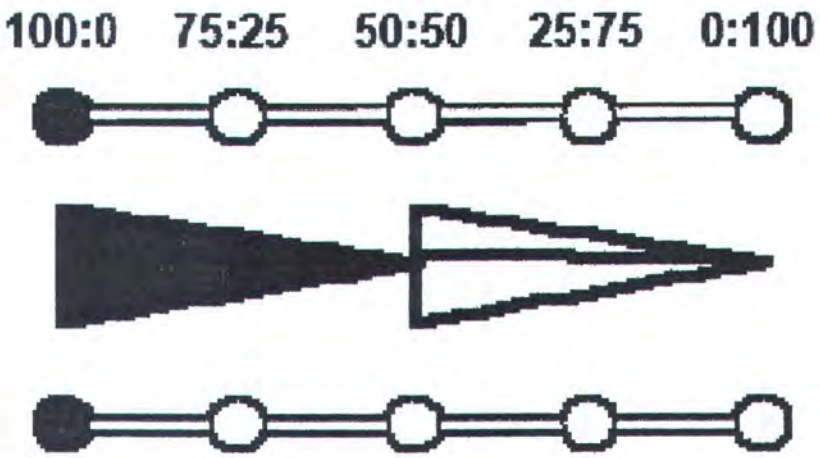


Gambar 5. Rotasi pada *basic skeletal animation*
a. Keadaan pada posisi awal
b. Keadaan setelah rotasi

Pada gambar 5 di atas, pemasangan verteks pada tulang digambarkan oleh warna yang sama antara verteks dan tulang. Dapat dilihat bahwa pada verteks yang berada tepat di atas sebuah sendi, pemasangan verteks pada tulang menjadi tidak jelas. Ketika terjadi rotasi, verteks yang berada tepat di atas sebuah sendi mengalami perentangan dan pemampatan *mesh* secara berlebihan. Untuk mengatasi permasalahan ini, dapat dipakai sebuah metode yang bernama *vertex blending*.

Dengan *vertex blending*, dapat dihasilkan suatu bentuk kulit yang halus dengan menggunakan kerangka yang tersegmentasi. Metode *vertex blending* ini dilakukan dengan cara memasangkan setiap verteks tidak hanya kepada satu

tulang saja, tetapi beberapa tulang yang berdekatan dengan bobot pemasangan tertentu. Bobot pemasangan ini dinamakan *vertex weight* yang harus berjumlah total 100%. Contoh pemasangan *vertex weight*:



Gambar 6. Contoh pemasangan *vertex weight*

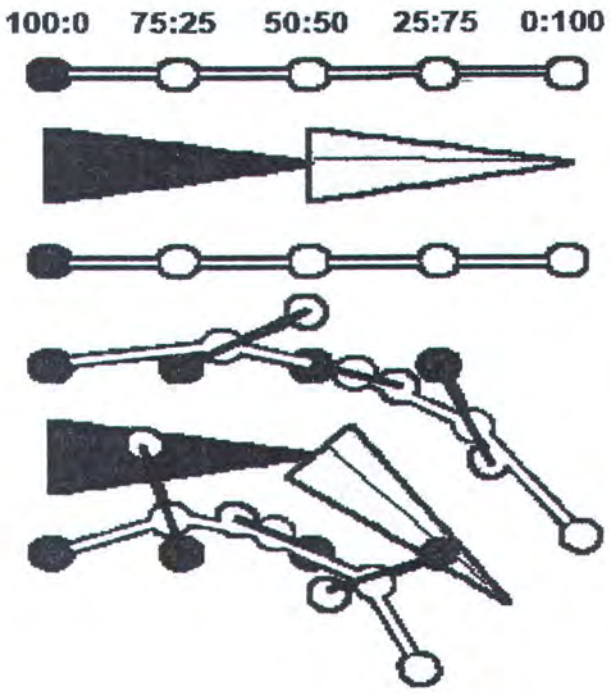
Pada gambar 6 di atas, dapat dilihat bahwa untuk setiap verteks, pemasangan terjadi pada setiap tulang *parent* dan *child* dengan persentase seperti yang telah dituliskan di atas setiap verteks. Pemasangan *vertex weight* ini dapat dilakukan pada waktu pembuatan model dengan menggunakan perangkat lunak *3D modeler* yang mempunyai fasilitas *vertex weight* seperti 3D Studio Max, Maya, dan lain-lain. Walaupun begitu, ada juga suatu metode heuristic untuk menghitung *vertex weight* berdasarkan pemasangan verteks ke tulang [07]. Kemudian, setelah terjadi sebuah rotasi, kedudukan akhir verteks adalah rata-rata kedudukan setiap verteks berdasarkan setiap tulang yang mempengaruhinya dengan bobot *vertex weight* terhadap tulang tersebut. Apabila terdapat *b* buah tulang pada sebuah kerangka, dan setiap verteks mempunyai *vertex weight*

sebesar w untuk setiap tulang pada kerangka, maka proses *blending* apabila terjadi rotasi dapat dinyatakan sebagai berikut:

$$v' = \sum_{i=0}^{b-1} w_i F(j_i) A(j_i)^{-1} v \tag{28}$$

Persamaan (28) di atas hampir sama dengan persamaan (3) untuk mencari posisi akhir verteks pada *basic skeletal animation*. Perbedaannya adalah, posisi akhir verteks ditentukan oleh total rata-rata posisi verteks berdasarkan pengaruh tulang i dengan bobot sebesar w_i. Sebagai catatan bahwa untuk sebagian besar dari nilai w_i untuk suatu verteks tertentu akan bernilai nol (0) karena tulang akan berada terlalu jauh dari verteks tersebut sehingga tidak terlalu mempengaruhi verteks ketika terjadi rotasi.

Penggunaan metode *vertex blending* ini dapat dilihat pada gambar berikut ini.



Gambar 7. Perataan verteks pada Vertex Blending

Pada gambar 7 di atas, dapat dilihat bahwa verteks dengan warna putih (posisi akhir verteks) adalah hasil rata-rata dari sebuah verteks dengan *vertex weight* masing-masing tulang abu-abu dan hitam. Hasil posisi akhir sementara verteks relatif terhadap tulang pasangan digambarkan dalam verteks yang berwarna sama dengan tulang pasangan yang mempengaruhinya. Hasil akhir dari posisi verteks hasil *vertex blending* (berwarna putih) merupakan hasil rata-rata antara dua verteks hitam dan abu-abu yang merupakan posisi akhir verteks relatif terhadap tulang masing-masing.

Ada satu kelemahan *vertex blending* yaitu apabila terjadi rotasi sebesar 180° terhadap sumbu rotasi tulang itu sendiri, maka akan terjadi suatu artifak yang disebut "*twisting elbow*" atau "*candy wrapper*" [07]. Artifak itu sendiri berarti sbuah distorsi atau kesalahan pada sebuah observasi[3]. Artifak ini dan pemecahannya akan dijelaskan lebih lanjut pada bagian subbab *bones blending* setelah ini. Walaupun masih mempunyai kelemahan, metode *vertex blending* ini sudah banyak digunakan dalam membuat animasi *humanoid* tiga dimensi secara *real time*, terutama pada permainan komputer. Hal ini karena *vertex blending* menghasilkan kondisi postur model berkerangka yang lebih baik daripada *basic skeletal animation* dengan hanya membutuhkan perhitungan yang sedikit lebih banyak. Sedangkan artifak "*twisting elbow*" juga jarang muncul karena tidak banyak terjadi rotasi sebesar 180° dengan sumbu tulang tempat penempatan *mesh.*

[3] http://en.wikipedia.org/wiki/Artifact

blending yang bernama *Spherical Blend Skinning*. Dengan metode ini rata-rata akan dilakukan bukan pada verteks, tetapi pada rotasi tulang. Metode ini dilakukan dengan cara melakukan rata-rata pada setiap posisi rotasi satu demi satu, bukan sekaligus secara keseluruhan. Menggabungkan dua buah rotasi dapat dengan mudah dilakukan dengan metode *SLERP*. Permasalahan yang muncul sekarang adalah bagaimana mencari nilai parameter *t* sebagai input dari *SLERP*.

Secara intuitif, *t* harus dihubungkan dengan jarak sepanjang tulang pada posisi kerangka referensi. Untuk menentukan jarak sepanjang tulang, digunakan matriks $A(j)^{-1}$ untuk mentransformasikan verteks dari posisi referensi ke posisi ternormalisasi, dengan sendi *j* dipusatkan pada titik pangkal koordinat. Jika sendi berikutnya adalah *k* dan *p(k)=j*, maka bagian tulang ditentukan oleh *R(k)*, yaitu komponen translasi dari *R(k)*. Vektor ini lalu dijadikan sebuah vektor unit dan dinotasikan dengan *n*. Kemudian dari verteks ternormalisasi sepanjang tulang dapat didefinisikan sebagai jarak dari bidang dengan garis normal *n* yang mengandung titik pangkal koordinat. Persamaan ini dapat dihitung sebagai perkalian skalar:

$$t' = v_0 \bullet n, v_0 = A(j)^{-1} v \tag{29}$$

dimana v_0 adalah versi *v* yang sudah ternormalisasi. Untuk membuktikan bahwa *t'* adalah jarak verteks ke bidang, anggap *nn*=1, maka:

$$(v_0 - t'n)n = v_0 n - t' = v_0 n - v_0 n = 0 \tag{30}$$

Nilai *t'* akan positif untuk titik pada arah *n*, nol untuk titik yang berada pada bidang, dan negatif untuk titik pada arah *–n*. Jika diasumsikan bahwa terdapat dua nilai:

- t_{min} = Nilai t' dimana hanya rotasi dari tulang interpolasi *parent* yang diterapkan.
- t_{max} = Nilai t' dimana hanya rotasi dari tulang interpolasi *child* yang diterapkan.

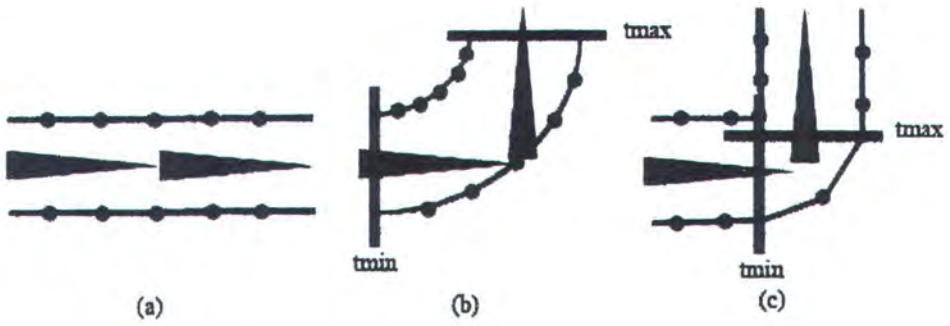
Untuk menentukan parameter interpolasi t adalah:

$$t = \frac{t' - t_{\min}}{t_{\max} - t_{\min}} \tag{31}$$

Parameter interpolasi t tidak akan berubah walaupun terjadi transformasi pada verteks karena parameter ini hanya ditentukan oleh posisi awal verteks pada posisi referensi. Karena itu parameter t ini hendaknya ditentukan hanya setelah *file* model *3D* dimuat pada aplikasi dan hasilnya disimpan pada *memory* agar dapat dipakai untuk seterusnya.

Tetapi untuk mencari parameter t tersebut, masih terdapat dua (2) permasalahan dalam menentukan beberapa konstanta yang akan dipakai dalam persamaan (31) di atas.

Permasalahan pertama adalah penentuan konstanta t_{max} dan t_{min}. Konstanta t_{max} dan t_{min} diperlukan untuk menentukan batas interpolasi transformasi. Konstanta ini ditentukan secara manual pada aplikasi bergantung kepada panjang tulang. Apabila konstanta t_{max} dan t_{min} ditentukan dengan nilai yang sangat besar maka verteks yang terkena interpolasi juga akan semakin banyak karena batas interpolasi terlalu lebar. Sedangkan apabila konstanta t_{max} dan t_{min} ditentukan dengan nilai yang sangat kecil maka verteks yang terkena interpolasi juga menjadi sedikit karena batas interpolasi terlalu sempit. Untuk lebih jelasnya dapat dilihat pada gambar di bawah ini.

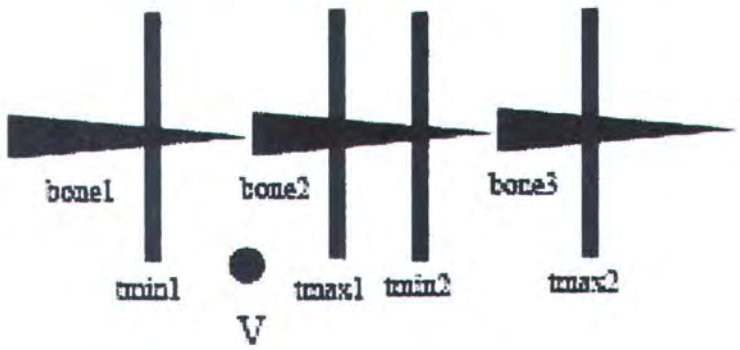


Gambar 9. Contoh penentuan t_{min} dan t_{max}
(a) Kondisi awal
(b) Rotasi dengan batas t_{max} dan t_{min} besar
(c) Rotasi dengan batas t_{max} dan t_{min} kecil

Apabila batas t_{max} dan t_{min} terlalu besar, maka hampir semua verteks pada tulang tersebut akan masuk ke dalam batas interpolasi. Hal ini akan menyebabkan deformasi kulit menjadi lebih bundar daripada yang seharusnya. Sedangkan apabila batas t_{max} dan t_{min} terlalu kecil, maka hanya sedikit verteks yang akan masuk ke dalam batas interpolasi. Bahkan apabila batas t_{max} dan t_{min} sangat kecil dan mendekati $t_{max} = t_{min}$ (batas t_{max} dan t_{min} berada tepat pada sendi di tengah) maka yang akan terjadi sama dengan *basic skeletal animation*. Hal ini terjadi karena tidak akan ada verteks yang masuk dalam batas interpolasi sehingga setiap verteks hanya akan dipengaruhi oleh transformasi dari tulang pasangannya.

Permasalahan kedua adalah bagaimana menentukan tulang kedua yang akan dipakai untuk menempatkan t_{max} atau t_{min}. Seperti yang telah dijelaskan, dalam *bones blending* sebuah verteks dipasangkan pada sebuah tulang. Sedangkan tulang itu sendiri berada pada sebuah pohon kerangka dalam bentuk sebuah *node* sendi. Karena itu, untuk setiap tulang, akan terdapat tulang *parent* (kecuali tulang pada *root joint*) dan tulang *child* (kecuali tulang pada ujung kerangka). Dalam

menentukan t_{max} dan t_{min}, diperlukan satu tulang untuk masing-masing konstanta (atau dua tulang untuk keduanya). Sedangkan tulang yang dapat dipakai ada tiga (3), yaitu: tulang pasangan, tulang *parent* dari tulang pasangan, dan tulang *child* dari tulang pasangan. Salah satu tulang yang pasti dipakai dalam menentukan t_{max} dan t_{min} adalah tulang pasangan verteks. Sedangkan untuk tulang kedua dapat ditentukan dari jarak verteks relatif terhadap kedua tulang yang tersisa (tulang *parent* dan *child*).



Gambar 10. Penentuan tulang kedua

Misal, pada gambar 9 di atas dapat dilihat posisi sebuah verteks *V* pada sebuah model kerangka. Verteks ini dipasangkan pada tulang *bone2*. Apabila tulang kedua yang dipilih adalah tulang *bone3*, maka pada *bone2* akan ditempatkan konstanta t_{min} (menjadi tulang interpolasi *parent*) sedangkan pada *bone3* akan ditempatkan konstanta t_{max} (menjadi tulang interpolasi *child*). Pada perhitungan parameter *t*, verteks *V* ini akan berada di luar batas interpolasi. Sehingga hanya transformasi *bone2* yang akan diimplementasikan pada verteks ini. Sedangkan apabila tulang kedua yang dipilih adalah tulang *bone1*, maka pada *bone1* akan ditempatkan konstanta t_{min} (menjadi tulang interpolasi *parent*)

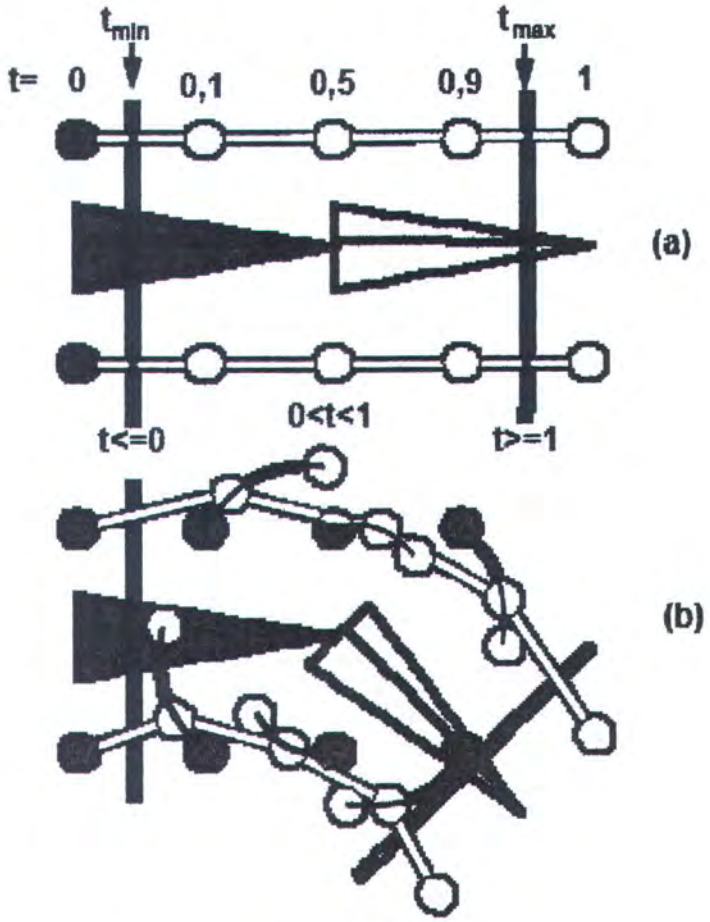
sedangkan pada *bone2* akan ditempatkan konstanta t_{max} (menjadi tulang interpolasi *child*). Posisi verteks *V* akan berada pada batas interpolasi. Sehingga transformasi yang terjadi pada verteks adalah hasil interpolasi antara transformasi *bone1* dan *bone2*. Tetapi dalam model sendiri umumnya suatu verteks sudah dipasangkan kepada tulang yang paling dekat dari verteks tersebut.

Setelah parameter interpolasi *t* diketahui, maka selanjutnya akan dihitung transformasi akhir verteks dari interpolasi antara transformasi tulang pertama dan transformasi tulang kedua dengan parameter interpolasi *t*. Transformasi akhir didapatkan sama seperti pada *basic skeletal animation.* Kecuali, untuk matriks *T(j)* yang dipakai adalah matriks hasil interpolasi. Interpolasi ini dapat dilakukan dengan berbagai pendekatan seperti *linear interpolation*, *spherical linear interpolation*, dan lain-lain. Mengingat transformasi yang terjadi pada verteks hanya berupa rotasi, maka pendekatan yang sebaiknya dipakai adalah *spherical linear interpolation (SLERP)* karena jalur interpolasi *SLERP* berupa lingkaran (sama seperti jalur pada rotasi).

Untuk verteks dengan $t<0$, maka transformasi yang akan diimplementasikan pada verteks adalah transformasi tulang interpolasi *parent*. Untuk verteks dengan $t>1$ akan diimplementasikan transformasi tulang interpolasi *child*. Sedangkan untuk $0>t>1$, maka transformasi yang akan diimplementasikan pada verteks adalah hasil interpolasi antara transformasi *parent* dan *child* dengan parameter *t*.

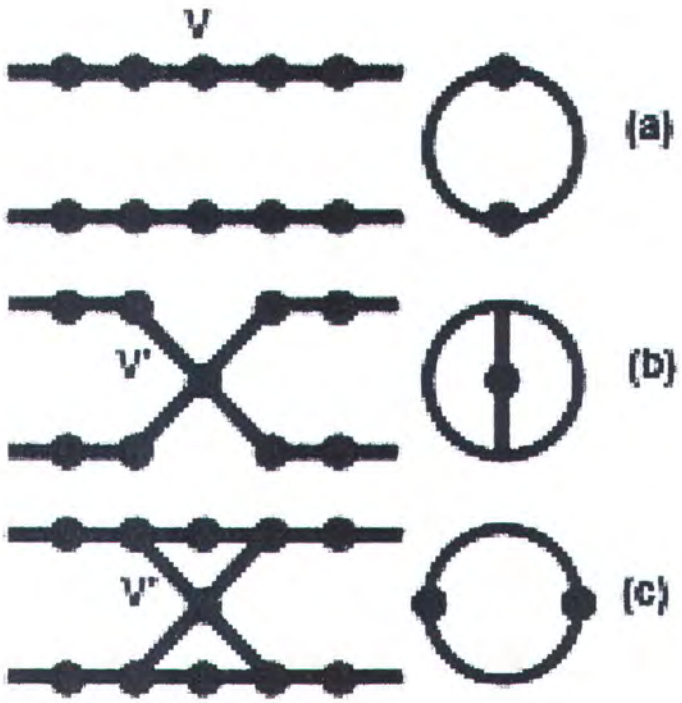
Dengan merata-rata pada level tulang, maka permasalahan penumpukan *digital skin* (kulit digital) tidak akan terjadi karena jarak antara verteks dan tulang

yang bersangkutan akan sama walaupun dilakukan transformasi. Untuk lebih jelasnya, dapat dilihat pada contoh gambar berikut:



Gambar 11. Bones Blending
a) Sebelum transformasi
b) Setelah transformasi

Untuk melihat perbandingan antara *verteks blending* dengan *bones blending* pada rotasi "melintir" 180° dapat dilihat pada gambar berikut.



Gambar 12. Perbandingan vertex blending dengan bones blending
a) Posisi awal kerangka
b) vertex blending
c) bones blending

Dapat dilihat pada gambar 12 di atas bahwa dengan metode bones blending, artifak *"twisting elbow"* atau *"candy wrapper"* tidak tampak.

BAB III
PERANCANGAN PERANGKAT LUNAK

BAB III
PERANCANGAN PERANGKAT LUNAK

Dalam bab ini akan dibahas perancangan dari perangkat lunak yang dibangun dalam tugas akhir ini. Perancangan perangkat lunak terbagi menjadi dua bagian, yaitu perancangan proses yang mencangkup penjelasan mengenai perancangan beberapa metode yang diperlukan dalam Tugas Akhir ini seperti *skeletal animation, quaternion, Sherical Linear intERPolation (SLERP), vertex blending*, dan *bones blending* serta perancangan antarmuka.

3.1. Perancangan Proses

Pada bagian ini diuraikan tentang perancangan proses untuk setiap proses yang akan dijalankan dalam perangkat lunak. Desain dari proses dijelaskan dengan bagan yang menggambarkan alur proses sesuai dengan algoritma yang digunakan.

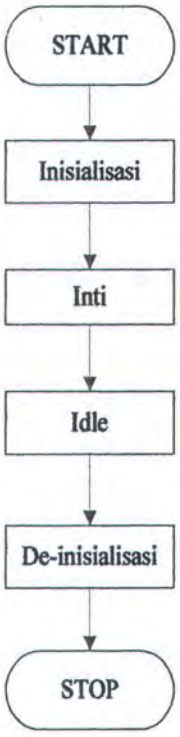
3.1.1. Perancangan Proses Umum

Berikut ini akan dijelaskan tentang perancangan proses yang terjadi dalam perangkat lunak secara garis besar. Proses yang terjadi secara umum adalah sebagai berikut:

1. Proses inisialisasi. Dalam proses ini akan dilakukan beberapa proses untuk mengatur hal-hal yang diperlukan untuk melakukan proses selanjutnya.

2. Proses inti. Dalam proses ini akan dilakukan beberapa proses yang merupakan bagian inti dari Tugas Akhir ini, yaitu proses animasi dan *skin deformation.*
3. Proses *idle*. Dalam proses ini, perangkat lunak tidak melakukan proses yang berarti selain hanya menunggu masukan dari antar muka perangkat lunak. Masukan yang akan dikenali adalah masukan untuk keluar dari program, masukan untuk mengubah mode *skin deformation*, dan masukan untuk memulai proses animasi.
4. Proses de-inisialisasi. Dalam proses ini perangkat lunak akan melakukan beberapa proses untuk mengakhiri sesi dan membersihkan objek-objek perangkat lunak yang telah dibuat dalam memori.

Proses-proses di atas dapat digambarkan dalam diagram alur berikut:



Gambar 13. Diagram Alur proses secara umum

Untuk setiap proses dalam proses umum di atas, akan dijelaskan dengan lebih terinci dalam subbab-subbab berikut ini:

3.1.1.1. Perancangan Proses Inisialisasi

Dalam proses ini akan melakukan beberapa inisialisasi untuk mengatur segala sesuatu yang diperlukan untuk melakukan proses inti dari Tugas Akhir ini. Alur proses ini adalah sebagai berikut:

Gambar 14. Diagram alur proses inisialisasi

Penjelasan diagram alur di atas adalah sebagai berikut:

- Inisialisasi Lingkungan Grafika

  Penginisialisasian lingkungan grafika yang dilakukan adalah penginisialisasian segala sesuatu yang akan digunakan untuk menampilkan tampilan *3D* seperti *viewport,* dunia *3D*, kamera, pencahayaan, dan lain-lain.

- Inisialisasi Data dan Objek Bergerak

  Data diinisialisasi dari model *3D* yang dibuat dari sebuah editor *3D* seperti *Milkshape, 3D Studio Max, Maya,* dan lain-lain yang telah dikonversi menjadi bentuk *.mesh* dan *.skeleton.*

  Data yang dikonversi nantinya akan berbentuk sebuah daftar tulang, sendi, *polygon* segitiga, *vertex* dari polygon segitiga, *vertex weight* dari setiap verteks, dan tekstur dari model tersebut. Apabila model yang dipakai mempunyai hubungan banyak tulang per verteks, maka *vertex weight* juga akan berbentuk daftar dimana terdapat data penunjuk ke tulang pasangan dan besar *vertex weight* yang akan bervariasi antara nol(0) sampai dengan satu(1). Sedangkan apabila hubungan tulang-verteks berupa satu (1) tulang per verteks, maka hanya akan terdapat satu (1) data *vertex weight* dalam daftar *vertex weight* yang berisi data penunjuk ke tulang pasangan dan besar *vertex weight* yang bernilai satu(1). Daftar tulang dan sendi akan berbentuk sebuah pohon hierarki dimana terdapat hubungan *parent* dan *child* antara tulang/sendi satu dengan tulang/sendi lainnya.

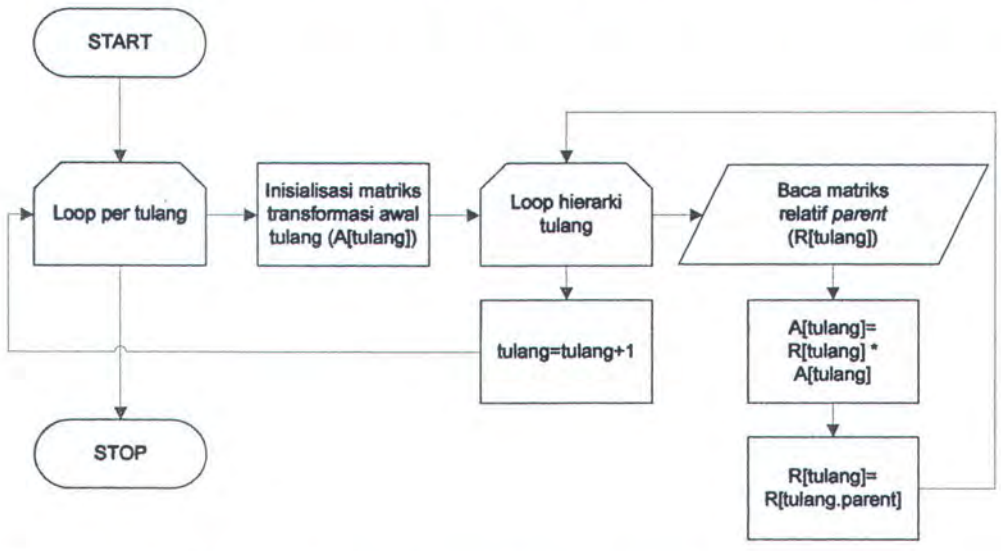
  Sesuai konvensi yang berlaku pada pemrograman *skeletal animation*, tulang tidak akan direpresentasikan sebagai tulang secara sesungguhnya, tetapi sebagai sendi *parent* dimana tulang tersebut tersambungkan dan panjang

tulang akan direpresentasikan dengan jarak antara sendi *parent* dan *child* dari tulang tersebut. Bagian kulit dari model akan terbentuk dari beberapa *polygon* segitiga dan tekstur dari kulit tersebut. Setiap *polygon* segitiga akan dibentuk oleh beberapa verteks. Karena setiap kulit akan dipasangkan ke tulang, maka daftar kulit secara tidak langsung akan berbentuk pohon hierarki mengikuti bentuk pohon dari daftar tulang dan sendi.

Selain data dari model, juga terdapat data yang dihitung berdasarkan data dari model *3D* dalam program seperti matriks transformasi dari posisi referensi yang akan digunakan untuk melakukan animasi. Proses penghitungan matriks transformasi ini dilakukan dengan persamaan (1) pada bab dasar teori yaitu:

$$A(j) = R(0)...R(p(j))R(j)$$

Diagram alir dari proses penghitungan matriks referensi ini adalah:



Gambar 15. Diagram alur proses penghitungan matriks posisi referensi

Dalam proses ini juga akan dilakukan inisialisasi data *t* dari bones blending. Data ini sebenarnya dapat diinisialisasi sekali saja dan tidak perlu dihitung berulang kali pada proses inti *Bones Blending*. Karena data ini tidak akan berubah kecuali batas t_{min} dan t_{max} juga diubah. Data masukan dari proses ini berasal dari data data model *3D* yang telah diinisialisasi.

Untuk menentukan variabel *t*, sebuah variabel yang berisi nilai dari jarak verteks yang sudah ternormalisasi sepanjang tulang (*t'*). Nilai variabel ini dapat dicari dengan persamaan (29):

$$t' = v_0 n, v_0 = A(j)^{-1} v$$

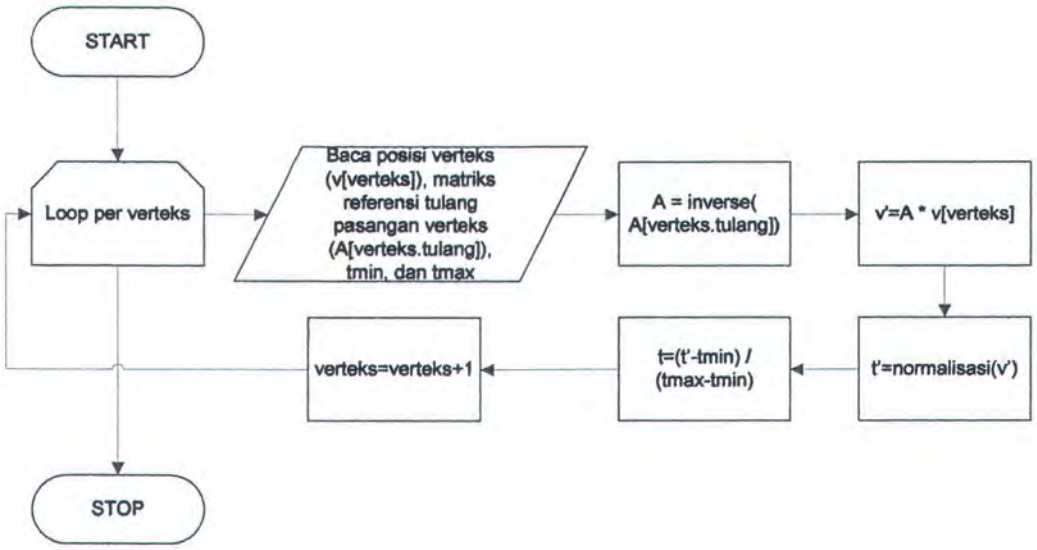
Variabel v_0 adalah sebuah verteks yang merupakan hasil normalisasi verteks yang akan dihitung posisi akhirnya. Variabel *n* adalah sebuah vektor unit dari matriks translasi tulang yang melakukan rotasi terhadap tulang *parent*.

Setelah nilai *t'* diketahui, dua buah konstanta t_{min} dan t_{max} dapat ditentukan dengan cari memilih nilai antara nol dan panjang tulang *parent* dan *child* dari verteks yang melakukan transformasi.

Dengan telah ditentukannya nilai-nilai *t'*, t_{min} , dan t_{max} maka nilai variabel *t* dapat dicari dengan persamaan (31):

$$t = \frac{t' - t_{\min}}{t_{\max} - t_{\min}}$$

Diagram alir dari proses mencari interpolan ini adalah:



Gambar 16. Diagram alur proses mencari interpolan *t*

Selain menentukan data dari suatu model, pada proses ini juga akan diinisialisasikan objek bergerak (*Entity*) yang data geometrinya berasal dari suatu *mesh*.

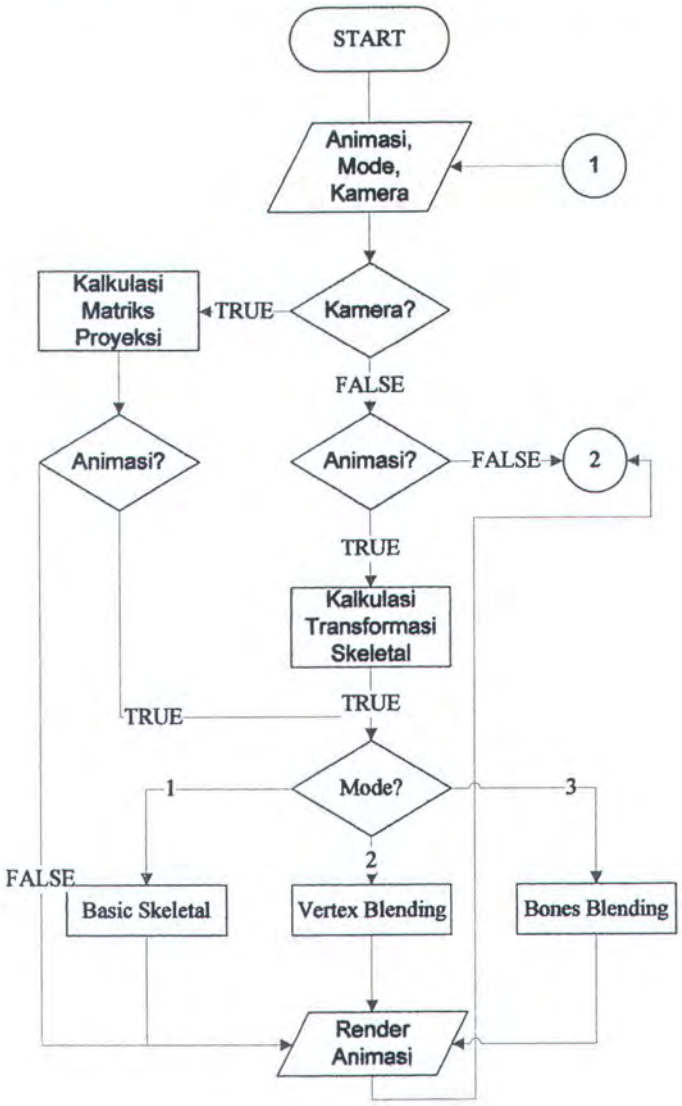
- Inisialisasi antar muka

Antar muka yang digunakan dalam perangkat lunak ini tidak akan berbentuk *GUI* (*Graphical User Interface*) karena tampilan perangkat lunak adalah dalam mode *full-screen* dan tidak dalam mode tampilan berjendela (*windowed*) seperti dalam *MFC* (*Microsoft Foundation Class*). Alasan dipilih mode *full-screen* adalah karena dalam mode tampilan berjendela, akan muncul pengurangan performa (seperti pengurangan *frame-rate*) yang akan menyebabkan tampilan animasi menjadi lebih lambat. Antar muka yang akan digunakan adalah masukan dari *keyboard*. Masukan dari *keyboard* akan berupa masukan untuk keluar dari perangkat lunak, masukan untuk memilih

mode metode *skin deformation* yang dipakai, dan masukan untuk memulai animasi.

3.1.1.2. Perancangan Proses Inti

Dalam proses ini, akan dilakukan segala inti dari perangkat lunak ini meliputi perhitungan *skin deformation*, aplikasi *skin deformation* pada model, animasi, dan *rendering*. Diagram alur dari proses inti adalah sebagai berikut:



Gambar 17. Diagram alur proses inti

Penjelasan diagram alur di atas adalah sebagai berikut:

- Input Animasi, Mode, dan Kamera

Animasi adalah sebuah variabel *boolean* yang digunakan untuk mengetahui apakah pengguna memilih untuk menjalankan aplikasi atau tidak. Sedangkan mode adalah sebuah variabel integer untuk memilih mode *skin deformation* yang digunakan. Mode=1 berarti pengguna memilih *basic skeletal*, mode=2 berarti pengguna memilih *vertex blending*, dan mode=3 berarti pengguna memilih *bones blending*. Nilai kedua variabel tersebut didapatkan dari proses Idle (referensi proses 1).

Apabila animasi=*false*, maka perangkat lunak akan kembali menjalankan proses Idle (referensi proses 2). Apabila animasi=*true*, maka perangkat lunak akan menjalankan proses kalkulasi transformasi *skeletal*.

Kamera adalah sebuah variabel yang menentukan apakah ada transformasi dari kamera yang akan mengubah tampilan berdasarkan sudut pandang kamera. Sama seperti kedua variabel sebelumnya, variabel ini juga didapatkan dari proses Idle.

- Kalkulasi Matriks Proyeksi

Matriks proyeksi didapatkan dari transformasi kamera terhadap koordinat dunia. Matriks ini dinamakan *view projection matrix* (matriks proyeksi pandangan) yang akan diimplementasikan untuk mengubah tampilan berdasarkan sudut pandang kamera.

Setelah selesai melakukan kalkulasi matriks ini, proses akan berjalan untuk mengenali apakah ada animasi atau tidak. Apabila variabel animasi

Setelah selesai melakukan kalkulasi matriks ini, proses akan berjalan untuk mengenali apakah ada animasi atau tidak. Apabila variabel animasi adalah *true* , maka proses akan menjalankan proses kalkulasi transformasi skeletal. Sedangkan apabila *false,* maka proses akan mengimplementasikan matriks proyeksi pandangan terhadap tampilan sebelumnya.

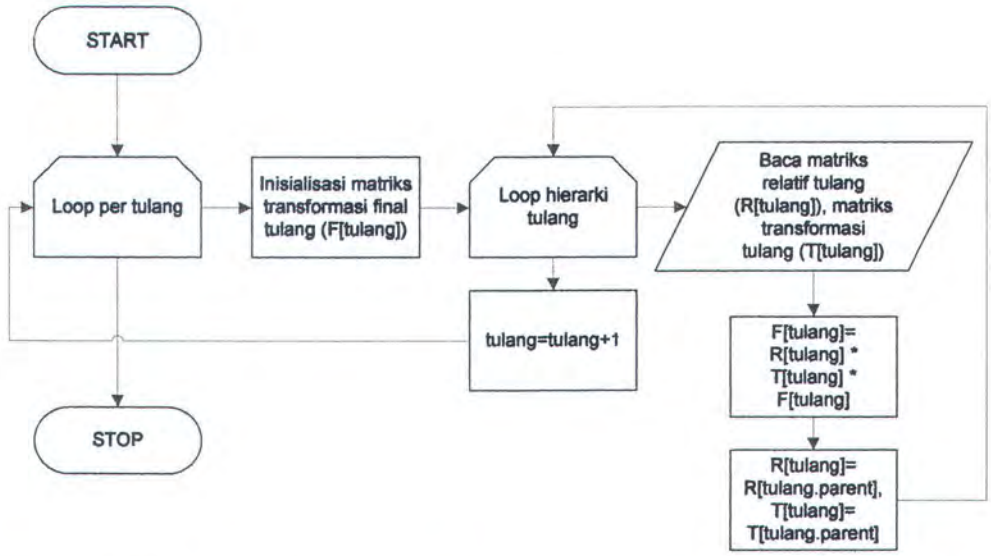
- Kalkulasi Transformasi Skeletal

Proses ini adalah proses untuk menghitung matriks transformasi dari setiap tulang dan sendi yang terdapat pada kerangka ketika terjadi transformasi yaitu matriks transformasi final. Matriks transformasi final adalah matriks transformasi dari transformasi yang sesungguhnya terjadi pada suatu sendi atau tulang. Matriks final ini dihitung berdasarkan persamaan (2):

$$F(j) = F(p(j))R(j)T(j)$$
$$F(0) = R(0)T(0)$$

Matriks T adalah matriks transformasi lokal yang terjadi pada sendi itu terhadap koordinat ruang lokal sendi. Matriks transformasi ini diperoleh dari masukan transformasi animasi yang ditentukan secara langsung pada perangkat lunak.

Diagram alir dari proses mencari matriks transformasi final ini adalah:



Gambar 18. Diagram alur proses penghitungan matriks transformasi final

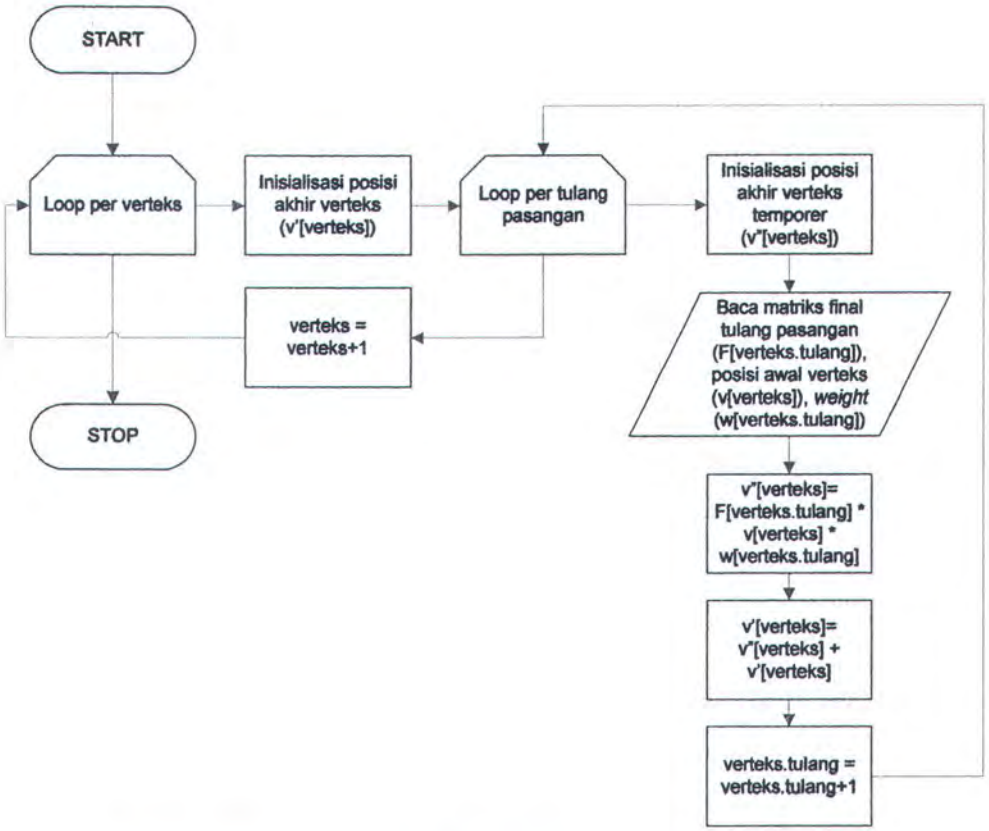
Setelah melakukan penghitungan transformasi pada kerangka, maka selanjutnya dilakukan pemilihan mode *skin deformation* yang akan dipakai. Apabila mode=1, maka perangkat lunak akan menjalankan proses *basic skeletal.* Apabila mode=2, maka perangkat lunak akan menjalankan proses *vertex blending.* Apabila mode=3, maka perangkat lunak akan menjalankan proses *bones blending.*

- Basic Skeletal

Dalam proses ini, dilakukan penghitungan transformasi yang terjadi pada setiap verteks berdasarkan mode *basic skeletal animation.* Penghitungan posisi verteks yang telah ditransformasikan adalah dengan menggunakan persamaan (3):

$$v' = F(j)A(j)^{-1}v$$

Pada persamaan (3) di atas, terdapat invers matriks absolut dari sendi tempat verteks dipasangkan. Sedangkan matriks yang telah diketahui nilainya



Gambar 20. Diagram alur proses *vertex blending*

- Bones Blending

Dalam proses ini, dilakukan penghitungan transformasi yang terjadi pada setiap verteks berdasarkan mode *bones blending*. Langkah pertama proses ini adalah melakukan perhitungan interpolasi dengan *SLERP* terhadap rotasi lokal (matriks transformasi *T(j)*) yang terjadi pada tulang *parent* dan tulang *child*.

Perhitungan interpolasi dengan menggunakan *SLERP* dilakukan dengan persamaan (26) :

$$SLERP(t; q_0, q_1) = \frac{\sin((1-t)\Phi) + \sin(t\Phi)q_1}{\sin(\Phi)}$$

Dimana q_0 adalah rotasi pada tulang *parent*, dan q_1 adalah rotasi pada tulang *child*. Untuk t=0, maka matriks transformasi rotasi lokal dari verteks sama dengan matriks transformasi rotasi lokal dari tulang *parent*. Sebaliknya, untuk t=1, maka matriks transformasi rotasi lokal dari verteks sama dengan matriks transformasi rotasi lokal dari tulang *child*.

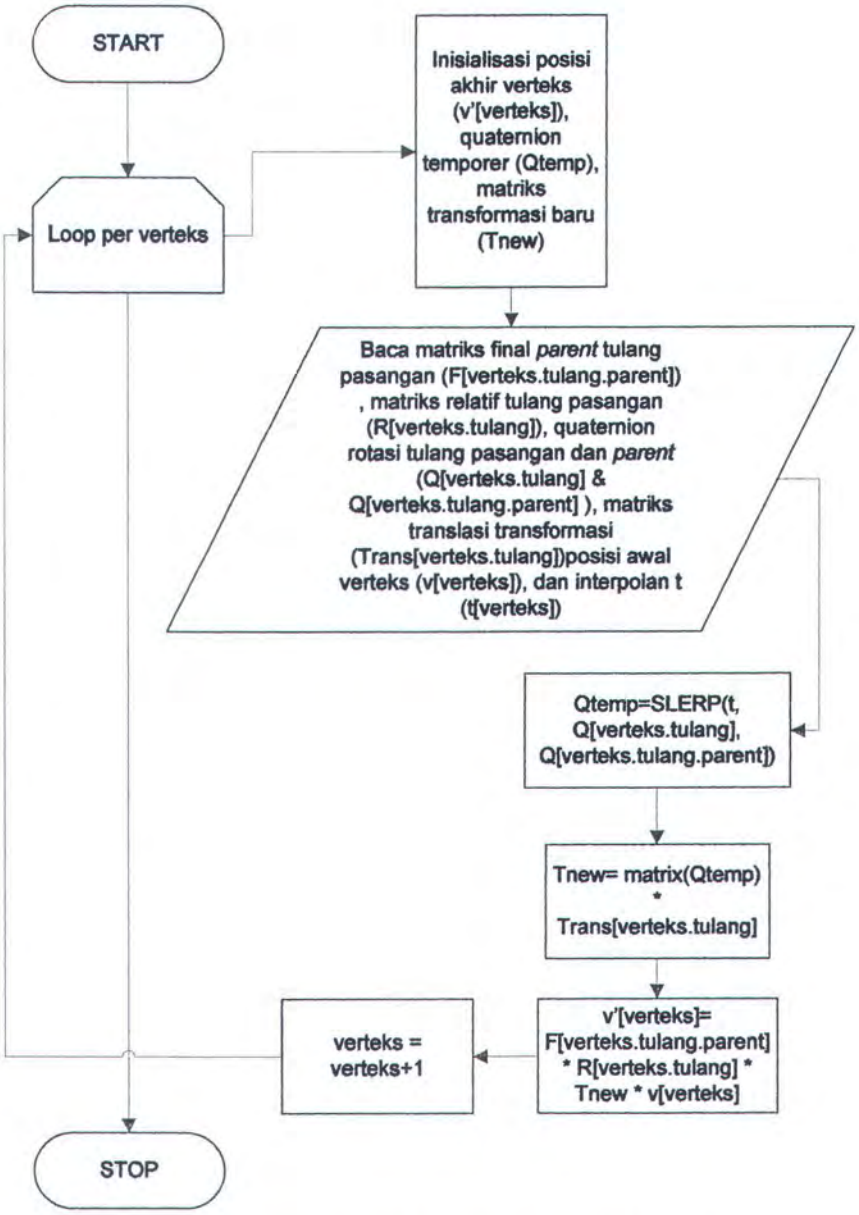
Matriks transformasi rotasi lokal yang dihasilkan akan menjadi matriks transformasi rotasi lokal verteks yang baru. Matriks ini nantinya akan digunakan untuk membuat matriks transformasi final *F(j)* yang baru dengan persamaan (2):

$$F(j) = F(p(j))R(j)T(j)$$
$$F(0) = R(0)T(0)$$

Setelah matriks transformasi final yang baru diketahui, maka matriks transformasi tersebut diaplikasikan untuk menentukan posisi akhir verteks dengan persamaan (3):

$$v' = F(j)A(j)^{-1}v$$

Diagram alir proses *bones blending* di atas adalah sebagai berikut:



Gambar 21. Diagram alur proses *bones blending*

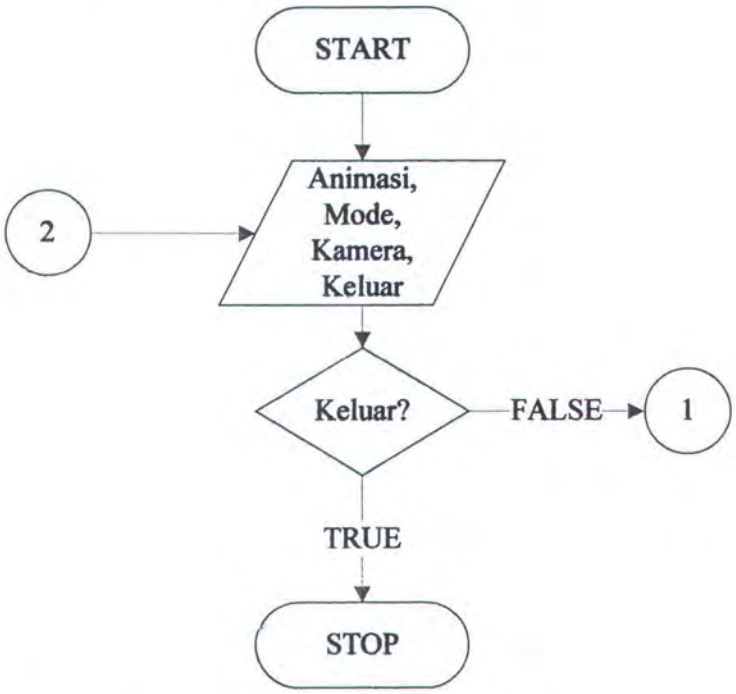
- Output Render Animasi

Setelah seluruh penghitungan dilakukan, maka tampilan akhir dari posisi model yang melakukan transformasi telah diketahui. Seluruh hasil

penghitungan dan posisi dalam koordinat *3D* selanjutnya ditampilkan pada layar oleh *rendering API.*

3.1.1.3. Perancangan Proses Idle

Dalam proses ini, perangkat lunak akan berada dalam kondisi *idle* dimana sebuah masukan diperlukan untuk kembali ke proses inti atau keluar dari program . Diagram alur dari proses *idle* adalah sebagai berikut:



Gambar 22. Diagram alur proses idle

Penjelasan diagram alur di atas adalah sebagai berikut:

- Input Animasi, Mode, Kamera, dan Keluar

  Pada proses ini, perangkat lunak akan menunggu masukan antara lain variabel *boolean* animasi dan keluar dan variabel *integer* mode. Variabel animasi digunakan untuk mengetahui apakah animasi model dipilih. Variabel keluar digunakan untuk mengetahui apakah pengguna ingin keluar dari

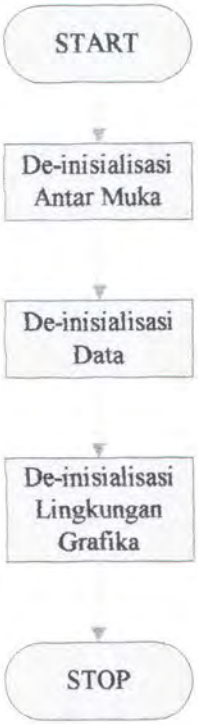
program. Variabel mode digunakan untuk memilih mode *skin deformation* yang diimplementasikan pada model. Mode=1 berarti pengguna memilih *basic skeletal*, mode=2 berarti pengguna memilih *vertex blending*, dan mode=3 berarti pengguna memilih *bones blending*.

Proses ini berjalan secara *real-time* sehingga setiap pemasukan nilai variabel akan selalu diperiksa secara rekursif.

Apabila variabel keluar bernilai *false*, maka proses akan beralih ke proses inti (referensi proses 1). Sedangkan apabila pada proses inti nilai variabel animasi adalah *false*, maka proses akan kembali lagi ke proses *idle* (referensi proses 2).

3.1.1.4. Perancangan Proses De-inisialisasi

Dalam proses ini, perangkat lunak akan mulai melakukan penghapusan dan deinisialisasi objek-objek perangkat lunak dari memori. Diagram alur dari proses de-inisialisasi adalah sebagai berikut:



Gambar 23. Diagram alur proses de-inisialisasi

Penjelasan diagram alur di atas adalah sebagai berikut:

- De-inisialisasi Antar Muka

  Pada proses ini dilakukan penghapusan dan deinisialisasi segala objek *buffer* yang digunakan untuk membaca inputan dari keyboard dan tetikus.

- De-inisialisasi Data

  Pada proses ini dilakukan penghapusan dan deinisialisasi segala objek data model *3D* dari memori. Perangkat lunak ini menggunakan model hanya untuk mendapatkan data dan tidak mengubah data tersebut secara permanen pada file model. Jadi tidak ada proses penyimpanan data dari perangkat lunak ke dalam file model. Data objek yang dihapus meliputi objek kerangka, tulang, sendi, polygon segitiga, verteks, matriks transformasi, dan lain-lain.

- De-inisialisasi Lingkungan Grafika

Pada proses ini dilakukan penghapusan dan deinisialisasi segala objek yang telah digunakan untuk menampilkan tampilan *3D* seperti *viewport*, kamera, pencahayaan, dan lain-lain.

3.2. Perancangan Antar Muka

Perangkat lunak akan berjalan pada mode *full screen*. Sehingga antar muka perangkat lunak tidak dalam bentuk *button*, *radio button*, *list*, dan objek *windows* lainnya.

Pemilihan mode *full screen* dalam perangkat lunak adalah karena dalam mode *full screen*, performa tampilan *3D* tidak akan begitu berkurang karena tidak adanya proses ekstra seperti bila memakai mode *windowed*.

Perangkat lunak mengenali masukannya dari tombol keyboard yang ditekan oleh pengguna. Secara garis besar, tombol-tombol yang diperlukan dalam perangkat lunak adalah:

- Tombol untuk keluar dari perangkat lunak dan kembali ke sistem utama.
- Tombol untuk memilih tampilan kamera
- Tombol untuk menggerakkan kamera
- Tombol untuk memulai animasi
- Tombol untuk memilih mode *skin deformation*

Selain itu juga akan dipakai masukan dari tetikus untuk menggerakkan tampilan kamera.

BAB IV
IMPLEMENTASI PERANGKAT LUNAK

BAB IV
IMPLEMENTASI PERANGKAT LUNAK

Setelah melalui proses perancangan, dilakukan pembuatan perangkat lunak. Pada bab ini akan dibahas tentang lingkungan dan tahap-tahap implementasi, yang dilanjutkan dengan penjelasan tentang masing-masing tahapan tersebut. Tetapi sebelumnya akan dijelaskan tentang sebuah komponen *Graphics Rendering Engine API* bernama OGRE 3D (*Object-oriented Graphics Rendering Engine 3D*) yang digunakan untuk membantu pembuatan Tugas Akhir ini.

4.1. OGRE 3D (Object-oriented Graphics Rendering Engine 3D)

OGRE 3D (*Object-oriented Graphics Rendering Engine 3D*) adalah sebuah *API open source* untuk mempermudah pembuatan lingkungan grafika 3 dimensi dan animasi real-time. Karena sifatnya *open source,* maka OGRE 3D dapat digunakan secara gratis menurut ketentuan LGPL.OGRE 3D dapat menggunakan model grafika DirectX ataupun OpenGL. *Source code* maupun *pre-compiled SDK* OGRE 3D dapat diambil secara gratis pada website official OGRE 3D yaitu http://www.ogre3d.org dengan versi *stable* terakhir sewaktu Tugas Akhir ini dibuat adalah OGRE 3D 1.0.6 .

OGRE 3D banyak digunakan dalam Tugas Akhir ini terutama dalam inisialisasi, de-inisialisasi, antar-muka, dan proses *rendering*. Untuk seterusnya dalam bab ini *API* OGRE 3D akan disebut hanya dengan *API* .

4.2. Tahap-tahap Implementasi

Implementasi perangkat lunak secara keseluruhan dibagi menjadi dua (2) dengan urutan sebagai berikut :

1. Implementasi proses, yaitu pembuatan perangkat lunak untuk masing-masing proses yang ada dalam sistem. Implementasi proses terdiri dari :
   - Implementasi proses inisialisasi
   - Implementasi proses inti
   - Implementasi proses *idle*
   - Implementasi proses de-inisialisasi
2. Implementasi antarmuka, yaitu pembuatan antarmuka untuk sistem secara keseluruhan

Masing-masing tahap implementasi ini akan dibahas pada beberapa subbab berikut.

4.3. Implementasi Proses

Pada sub bab ini akan dijelaskan mengenai implementasi dari perancangan proses yang telah dibuat pada bab III. Adapun proses-proses yang akan diimplementasikan adalah :

- Implementasi proses inisialisasi yang meliputi :
  - Implementasi subproses Inisialisasi Data
  - Implementasi subproses Inisialisasi Lingkungan Grafika
  - Implementasi subproses Inisialisasi Antar Muka
- Implementasi proses inti yang meliputi :

- Implementasi subproses Input Animasi, Mode dan Kamera
- Implementasi subproses Kalkulasi Matriks Proyeksi
- Implementasi subproses Kalkulasi Transformasi Skeletal
- Implementasi subproses Basic Skeletal
- Implementasi subproses Vertex Blending
- Implementasi subproses Bones Blending
- Implementasi subproses Output Render Animasi

▪ Implementasi proses *idle* yang meliputi:

- Implementasi subproses Input Animasi, Mode, Kamera, dan Keluar

▪ Implementasi proses de-inisialisasi yang meliputi:

- Implementasi subproses De-inisialisasi Antar Muka
- Implementasi subproses De-inisialisasi Data &
- Implementasi subproses De-inisialisasi Lingkungan Grafika

Implementasi subproses-subproses di atas akan dijelaskan lebih lanjut pada beberapa subbab berikut.

4.3.1. Implementasi Subproses Inisialisasi Lingkungan Grafika

Subproses ini sepenuhnya menggunakan bantuan dari *API*. Dalam *API*, objek-objek yang diinisialisasikan untuk membentuk suatu lingkungan grafika adalah:

- *Root*: Objek yang paling utama dari OGRE yang merupakan objek untuk mengkonfigurasikan sistem *rendering*. Objek ini adalah

tempat penempatan objek-objek yang lain. Objek ini harus dibuat pertama kali dan dihancurkan paling akhir.

- *RenderSystem*: Objek yang bertanggungjawab mengirim perintah *rendering* kepada DirectX (Direct3D) atau OpenGL.
- *SceneManager*: Objek yang bertanggungjawab isi dari suatu tampilan yang akan di*render*. Objek ini bertanggungjawab mengatur kamera, objek-objek 3D yang dinamis/bisa ditransformasikan (disebut *Entity*), pencahayaan dan material, dan geometri dunia (biasanya berupa objek statis).
- *ResourceManager*: Kelas dasar yang bertanggungjawab untuk mengambil data-data yang diperlukan oleh *API* seperti mesh, tekstur, dan lain-lain. Kelas ini akan diturunkan menjadi objek-objek spesifik berdasarkan kegunaannya masing-masing seperti *TextureManager, MeshManager,* dan lain-lain.
- *Mesh*: Objek yang merepresentasikan sebuah model nyata, kumpulan geometri yang disimpan tersendiri, dan kebanyakan relatif kecil terhadap dunia. Data *Mesh* didapatkan dari file berekstensi *.mesh* (dan *.skeleton* jika mempunyai kerangka) atau bisa dideklarasikan secara manual.
- *Entity*: Objek yang merupakan representasi objek bergerak (objek yang dapat ditransformasikan) dalam dunia. Objek ini mendapatkan data geometrinya dari objek *Mesh*.

- *Material*: Objek yang bertanggungjawab mengontrol bagaimana sebuah objek dalam suatu tampilan akan di*render*. Objek ini mengatur properti dasar permukaan seperti *reflectance, shininess, texture mapping,* dan lain-lain.
- *Overlay*: Objek yang dapat digunakan untuk menampilkan elemen-elemen *2D* atau *3D* di atas tampilan normal posisinya selalu tetap. Berguna untuk membuat tombol, panel, menu, dan lain-lain. Objek ini tidak diharuskan ada.
- *Skeleton, Bone, Animation*: Objek-objek yang berguna untuk membuat animasi skeletal (*skeletal animation*). Data-data yang diperlukan oleh objek-objek ini didapatkan dari *.skeleton* yang dibaca sekaligus ketika membaca suatu *mesh*.

Beberapa dari objek-objek ini tidak akan diinisialisasikan pada subproses ini, melainkan pada subproses setelah subproses ini, yaitu subproses Inisialisasi Data dan Objek Bergerak.

Pseudo-code dari subproses ini adalah:

```
//create root
Root* mRoot = new Root();

//tambah lokasi file untuk resource manager
ResourceGroupManager::getSingleton().addResourceLocation(
            archName, typeName, secName);

//choose scene manager
SceneManager* mSceneMgr = mRoot->getSceneManager(ST_GENERIC);

//create camera
Camera* mCamera = mSceneMgr->createCamera("PlayerCam");

//create viewport
Viewport* vp = mWindow->addViewport(mCamera);

//load resource
```

```
ResourceGroupManager::getSingleton().initialiseAllResourceGroups();

//set lighting
mSceneMgr->setAmbientLight( AMB_LIGHT );
Light *light = mSceneMgr->createLight( "Light1" );
light->setType( Light::LT_POINT );
light->setPosition( Vector3(250, 150, 250) );
light->setDiffuseColour( PT_DIFF_LIGHT );
light->setSpecularColour( PT_SPEC_LIGHT );
light->setVisible(false);

//create scene node entity
//scene node adalah tempat penempatan objek (camera, entity, dll) dalam tampilan
SceneNode *node = mSceneMgr->getRootSceneNode()->createChildSceneNode(
                        "ModelNode", Vector3(0,0,0) );

//create scene node camera
node = mSceneMgr->getRootSceneNode()->createChildSceneNode( "CamNode1",
                        Vector3( 0, 100, 300 ) );

//Pitch node
//Gambar hierarchy: Root Scene Manager --> CamNode1 --> PitchNode1(Kamera di-
attach di Pitchnode)
// Create the pitch node
//Posisi kamera 1 = di depan model (default)
node = node->createChildSceneNode( "PitchNode1" );
node->attachObject( mCamera );

// Posisi kamera 2 = di belakang model
// create the second camera node/pitch node
node = mSceneMgr->getRootSceneNode()->createChildSceneNode( "CamNode2",
                        Vector3( 0, 100, -300 ) );
quat=Quaternion(Degree(180), Vector3::UNIT_Y);
node->rotate(quat);
node = node->createChildSceneNode( "PitchNode2" );

// Posisi kamera 3 = di atas model
// create the second camera node/pitch node
node = mSceneMgr->getRootSceneNode()->createChildSceneNode( "CamNode3",
                        Vector3( 0, 500, 0 ) );
node = node->createChildSceneNode( "PitchNode3" );
quat=Quaternion(Degree(-90), Vector3::UNIT_X);
node->rotate(quat);

// Posisi kamera 4 = di kiri model
// create the second camera node/pitch node
node = mSceneMgr->getRootSceneNode()->createChildSceneNode( "CamNode4",
                        Vector3( 300, 100, 0 ) );
quat=Quaternion(Degree(90), Vector3::UNIT_Y);
node->rotate(quat);
node = node->createChildSceneNode( "PitchNode4" );

// Posisi kamera 5 = di kanan model
// create the second camera node/pitch node
node = mSceneMgr->getRootSceneNode()->createChildSceneNode( "CamNode5",
```

```
                                Vector3( -300, 100, 0 ) );
quat=Quaternion(Degree(-90), Vector3::UNIT_Y);
node->rotate(quat);
node = node->createChildSceneNode( "PitchNode5" );

// Posisi kamera 6 = melihat bahu model
// create the second camera node/pitch node
node = mSceneMgr->getRootSceneNode()->createChildSceneNode( "CamNode6",
                                Vector3( -30, 150, 50 ) );
node = node->createChildSceneNode( "PitchNode6" );

// Posisi kamera 7 = melihat siku model
 // create the second camera node/pitch node
node = mSceneMgr->getRootSceneNode()->createChildSceneNode( "CamNode7",
                                Vector3( -50, 200, 0 ) );
node = node->createChildSceneNode( "PitchNode7" );
quat=Quaternion(Degree(-90), Vector3::UNIT_X);
node->rotate(quat);

//Overlay
Overlay *TAOverlay = OverlayManager::getSingleton().getByName("TADemo");
TAOverlay->show();
```

4.3.2. Implementasi Subproses Inisialisasi Data dan Objek Bergerak

Subproses inisialisasi data dimulai dengan pengambilan data dari file model 3D yang berupa .mesh dan .skeleton yang merupakan data mesh dan skeleton dari model yang akan disimpan dalam memory. Subproses pengambilan data ini dilakukan oleh *API* . Data ini yang telah diambil kemudian akan disimpan ke dalam memory eksternal (seperti memory kartu grafis) oleh *API* untuk lebih mempersingkat jalur transfer data. Model *3D* yang akan diambil datanya ada dua (2) yaitu model dengan penempatan verteks ke tulang hanya satu (1) tulang untuk setiap verteks (*single weighted model*) dan model dengan penempatan verteks ke tulang adalah beberapa tulang untuk setiap verteks (*multi weighted model*). Maksimum tulang per verteks pada *multi weighted model* adalah tiga (3) buah tulang per verteks.

Sedangkan untuk model *Bones Blending*, dilakukan proses tersendiri yang tidak terdapat pada *API* untuk menentukan interpolan *t* dan tulang *parent* dan *child* dari verteks. Perhitungan ini dilakukan berdasarkan *single weighted model* yang telah diduplikasi. Metode ini dilakukan dengan menghitung interpolan *t* berdasarkan posisi setiap verteks dalam batas t_{min} dan t_{max}. Metode ini akan dilakukan secara rekursif terhadap setiap verteks dan akan menghasilkan penempatan tulang dan *weight* yang baru.

Pseudo-code dari subproses ini adalah:

```
//fungsi mengambil data model single weighted oleh API
Mesh Single = loadMesh(single);
//entity bernama "TASimpleBSA"
Entity *entity1 = mSceneMgr->createEntity( "TASimpleBSA", Single);

//fungsi mengambil data model multi weighted oleh API
Mesh Multi = loadMesh(multi);
//entity bernama "TASimpleVB"
Entity *entity2 = mSceneMgr->createEntity( "TASimpleVB", Multi);

//fungsi pembuatan mesh Bones Blending
//1. Duplikasi mesh single
Mesh  Bblend = Single.clone();
//entity bernama "TASimpleBB"
Entity *ent = mSceneMgr->createEntity( "TASimpleBB", Bblend);

//2. Menghitung interpolan t
float lowBound, hiBound;
Bone* boneArray = new Bone[4* Bblend.numVertices];
float*  tArray = new float[Bblend.numVertices];

for (int i=0;i<Bblend.numVertices;i++)
{
 Vector3 curVertex = Bblend.Vertices[i];
 Bone*    bone1 = Bblend->getBone(i*4); //List bone selalu 4 buah
 Bone* bone2, bone3;

 //Jika bukan root atau akhir hierarchy
 if (bone1.getParent()==0 || bone1.getChildrenNum()==0)
 {
  bone2 = bone1->getChild(0);

   //Balik bone1=bone2, bone2=bone3, bone1=parent(bone1)
  bone3 = bone2;
  bone2 = bone1;
  bone1 = bone1->getParent();
```

```
Matrix4 matr=bone1->_getBindingPoseInverseTransform(); //inv(A(bone1))
Matrix4 matr2=bone1->_getFullTransform();//F(bone1)
Matrix4 matr3=matr.inverse(); //inv(inv(A(bone1)) = A(bone1);

Vector3 vert=matr*curVertex;//A(bone1)*V(bone1)

Vector3 tUnit=bone2->getPosition(); //ambil matriks  translation dari R(bone2)
tUnit.normalise(); //Normalisasi tUnit

float hasil=vert.dotProduct(unitku);

//Cari tmin dan tmax
Real tmin=(((bone2->getPosition()).length())*(1.0f-(lowBound)));
Real tmax=((bone2->getPosition()).length())+(((bone3
      ->getPosition()).length())*hiBound);

//Cari t
tArray[i]=(hasil-tmin)/(tmax-tmin);

//Balik nilai ke posisi semula
bone3=bone1;
bone1=bone2;
bone2=bone3;
tArray[i]=1-tArray[i];

//0<t<1
tArray[i]=(tArray[i]<0.0)? 0.0f:tArray[i];
tArray[i]=(tArray[i]>1.0)? 1.0f:tArray[i];

boneArray [4*i]=bone1;
boneArray [(4*i)+1]=bone2;
boneArray [(4*i)+2]=0;
boneArray [(4*i)+3]=0;

 }//end if
}//end for

//Pasang entity "TASimpleBSA" sebagai entity awal
modelNode->getName()->attachObject( mSceneMgr->getEntity("TASimpleBSA") );
```

4.3.3. Implementasi Subproses Inisialisasi Antarmuka

Di dalam API terdapat sebuah objek yang berguna untuk membuat sebuah antar muka yang mengambil masukan dari keyboard dan tetikus. Nama dari objek ini adalah FrameListener. Sekali dibuat, objek ini akan selalu dijalankan di dalam *rendering loop*. Isi dari objek ini dapat ditentukan sendiri oleh *programmer* yang

memakai *API* untuk menentukan tombol apa saja yang akan dikenal. Sedangkan untuk menginisialisasikan objek ini hanya dengan dua baris saja.

Pseudo-code dari subproses ini adalah:

```
// Create the FrameListener
mFrameListener = new TAFrameListener(mWindow, mCamera, mSceneMgr,
                        vertices_temp,      normals_temp,      bone_indices_temp,
                        tfinal,vertex_count);
mRoot->addFrameListener(mFrameListener);
```

4.3.4. Implementasi Subproses Input Animasi, Mode dan Kamera

Pada subproses ini akan diperiksa apakah ada perubahan variabel Animasi, Mode, dan Kamera yang dikeluarkan oleh proses *idle*. Apabila terdapat perubahan posisi kamera, maka akan dilakukan subproses Kalkulasi Matriks Proyeksi. Sedangkan apabila tidak, maka akan diperiksa apakah variabel Animasi adalah *true* atau *false*. Apabila *true*, maka akan dilakukan subproses Kalkulasi Transformasi Skeletal. Sedangkan apabila *false*, maka akan menuju ke proses *idle*. Variabel Mode bernilai 1 sampai dengan 3 dimana 1 adalah mode "Basic Skeletal Animation", 2 adalah mode "Vertex Blending", dan 3 adalah mode "Bones Blending".

Subproses ini dilakukan secara automatis oleh *API* kecuali ketika terdapat animasi karena animasi ditentukan oleh *FrameListener*.

Pseudo-code dari subproses ini adalah:

```
//selain animasi, proses lainnya dilakukan oleh API
//animasi
else if (Animasi)
{
   KalkulasiTransformasiSkeletal();
}
```

4.3.5. Implementasi Subproses Kalkulasi Matriks Proyeksi

Dalam subproses ini penghitungan matriks proyeksi baru akan dilakukan oleh *API*. Setelah terbentuk matriks proyeksi baru, maka akan diperiksa apakah variabel Animasi bernilai *true* atau *false*. Apabila *true,* maka akan dilakukan subproses Kalkulasi Transformasi Skeletal. Sedangkan apabila *false*, maka akan menuju ke subproses Output Render Animasi. Sama seperti subproses sebelumnya, subproses ini semuanya dilakukan oleh *API* kecuali ketika mendeteksi animasi karena animasi ditentukan oleh *FrameListener*.

Pseudo-code dari subproses ini adalah:

```
//selain animasi, proses lainnya dilakukan oleh API
//animasi
else if (Animasi)
{
    KalkulasiTransformasiSkeletal();
}
```

4.3.6. Implementasi Subproses Kalkulasi Transformasi Skeletal

Dalam subproses ini pertama akan diperiksa mode apakah yang sedang aktif saat ini. Lalu akan diperiksa apakah entity model yang sekarang sesuai dengan mode yang bersangkutan. Jika tidak, maka entity model akan di-"detach" dari scene node model dan entity yang sesuai akan di-"attach". Kemudian dilakukan rotasi terhadap sebuah tulang (bone 1) pada kerangka. Rotasi mempunyai besar dari 0 sampai dengan 45 derajat dengan derajat pertambahan perframe adalah waktu yang telah lewat antara frame sekarang dan frame sebelumnya. Apabila lebih dari 45 derajat, maka akan ditentukan Animasi = *false*. Rotasi juga berdasarkan sumbu yang telah ditentukan. Apabila variabel mSumbuAnimasi yang didapatkan adalah 1, maka rotasi akan bersumbu Y,

sedangkan apabila yang didapatkan adalah 2, maka rotasi akan bersumbu Z. Setelah melakukan perubahan rotasi, maka penghitungan matriks transformasi skeletal baru akan dilakukan oleh *API*. Setelah terbentuk matriks transformasi skeletal baru, maka akan diperiksa mode apakah yang sedang aktif saat ini. Apabila bernilai 1, maka akan dilakukan subproses Basic Skeletal. Apabila bernilai 2, maka akan dilakukan subproses Vertex Blending. Sedangkan apabila bernilai 3, maka akan dilakukan subproses Bones Blending.

Pseudo-code dari subproses ini adalah:

```
if (Mode==1)
{
    if (ent!="TASimpleBSA")
    {
modelNode->getName()->detachAllObject();
modelNode->getName()->attachObject( mSceneMgr->getEntity("TASimpleBSA") );
    }
}
else if (Mode==2)
{
    if (ent!="TASimpleVB")
    {
modelNode->getName()->detachAllObject();
modelNode->getName()->attachObject( mSceneMgr->getEntity("TASimpleVB") );
    }
}
else
{
    if (ent!="TASimpleBB")
    {
modelNode->getName()->detachAllObject();
modelNode->getName()->attachObject( mSceneMgr->getEntity("TASimpleBB") );
    }
}


float sudut = timeSinceLastFrame();
mTotalSudutAnimasi += sudut;
if (mSumbuAnimasi==1) getBone("Bone 1")->
                rotate(Degree(mTotalAnimationAngle),Vector3::Y);
else if (mSumbuAnimasi==2) getBone("Bone 1")->
                rotate(Degree(mTotalAnimationAngle),Vector3::Z);

//penghitungan matriks transformasi skeletal dilakukan oleh API
CalculateSkeletalAnimation();
If (mTotalAnimationAngle>45)
```

```
{
   mTotalAnimationAngle=0;
   Animasi = false;
}

//Basic Skeletal
f (Mode==1)
{
    BasicSkeletal();
}

//Vertex Blending
else if (Mode==1)
{
    VertexBlending();
}

//Bones Blending
else if (Mode==1)
{
    BonesBlending();
}
```

4.3.7. Implementasi Subproses Basic Skeletal

Metode *basic skeletal animation* dan *vertex blending* sudah lama dikenal dalam dunia grafika *3D*. Karena itu, dalam *API* sudah terdapat metode *basic skeletal animation* dan *vertex blending*. Metode dalam *API* ini yang akan dipakai dalam Tugas Akhir ini. Setelah dilakukan metode penentuan verteks dengan metode *basic skeletal animation*, maka selanjutnya akan dilakukan proses *rendering* oleh *API*. Subproses ini semuanya dilakukan oleh *API* secara automatis.

Karena subproses ini sepenuhnya dilakukan oleh *API*, maka *pseudo-code* dari subproses ini tidak dapat.

4.3.8. Implementasi Subproses Vertex Blending

Metode *basic skeletal animation* dan *vertex blending* sudah lama dikenal odalam dunia grafika *3D*. Karena itu, dalam *API* sudah terdapat metode *basic skeletal animation* dan *vertex blending*. Metode dalam *API* ini yang akan dipakai

dalam Tugas Akhir ini. Setelah dilakukan metode penentuan verteks dengan metode *vertex blending*, maka selanjutnya akan dilakukan proses *rendering* oleh *API*. Karena dalam *API,* maksimum penempatan tulang per vertex adalah 4, maka dalam file model penempatan tulang per vertex maksimum harus 4.

Subproses ini semuanya dilakukan oleh *API* secara automatis.

Karena subproses ini sepenuhnya dilakukan oleh *API*, maka *pseudo-code* dari subproses ini tidak dapat ditentukan.

4.3.9. Implementasi Subproses Bones Blending

Implementasi subproses ini pertama dilakukan dengan mengambil matriks-matriks dari tulang-tulang dalam kerangka yang sudah yang sudah dianimasikan. Dalam *API*, matriks-matriks ini sudah disederhanakan menjadi sebuah matriks transformasi $M(j)$ dimana j adalah index tulang. Matriks $M(j)$ ini merupakan penyederhanaan dari $F(j).A(j)^{-1}$ dalam *basic skeletal animation.* Matriks inilah yang akan diimplementasikan kepada setiap verteks. Beberapa matriks komponen awal dari *skeletal animation* yang masih tetap ada dalam *API* adalah matriks transformasi total ($F(j)$), inverse matriks posisi referensi ($A(j)^{-1}$), matriks transformasi lokal tulang dalam posisi referensi ($R(j)$), dan sebuah penyederhanaan dari matriks $C(j)=R(j).T(j)$ yang merupakan matriks transformasi tulang yang melakukan animasi relatif terhadap *parent*. Selain itu, matriks $C(j)$ dan $R(j)$ dalam *API* dinyatakan oleh rotasi berupa *quaternion* dan translasi berupa vektor. Sehingga harus dikonversikan dalam bentuk matriks. Karena *bones blending* dilakukan pada matriks $T(j)$, maka pertama yang harus dilakukan adalah mengambil nilai matriks $T(j)$ dari matriks $C(j)$. Metode untuk melakukannya

adalah melakukan perkalian matriks $T(j)=R(j)^{-1}.C(j)$. Metode ini dilaksanakan secara rekursif untuk setiap bone.

Setelah nilai matriks *T(j)* diketahui, maka selanjutnya dilakukan proses *SLERP* dengan menggunakan matriks *T(j)*, matriks *T(parent j)* dimana *j* adalah tulang pasangan vertex tertentu dan interpolant *t* dari vertex tersebut. Hasil dari proses *SLERP* ini adalah sebuah matrix *T(j)* baru yang akan digunakan untuk membuat matriks *M(j)* yang baru. Lalu posisi dan normal dari verteks akan dikalikan dengan matriks *M(j)* yang baru ini. Metode ini dilaksanakan secara rekursif untuk setiap vertex.

Pseudo-code dari subproses ini adalah:

```
Entity* ent = mSceneMgr->getEntity("TASimpleBB");

//Data Bones Blending:
Matrix4* mFullMatrix = new Matrix4[mTotalBone]; //F(j)
Matrix4* mInitialMatrix = new Matrix4[mTotalBone]; //R(j)
Quaternion* mCurrQuaternion = new Quaternion[mTotalBone]; //quaternion dari T(j)
Vector3* mCurrTrans = new Vector3[mTotalBone]; //translasi dari T(j)
Matrix4* mBindPoseInverseMatrix = new Matrix4[mTotalBone]; //inv(A(j))
Matrix4* mLocalMatrix = new Matrix4[mTotalBone]; //C(j) = R(j).T(j)

Vector3* mDestPos= new Vector3[mVertexCount]; //posisi akhir vertex
Vector3* mDestNorm= new Vector3[mVertexCount]; //normal akhir vertex

for (unsigned int j=0;j<numBoneMat;j++) //Looping bone
{
        Bone* currBone= getBone(j);

        //Membuat transformasi R(j)
        Matrix4 relXform;
        relXform = Matrix4::IDENTITY;
        Matrix3 rot3x3;
        Quaternion relQuat = currBone->getInitialOrientation();
        relQuat.ToRotationMatrix(rot3x3);

        relXform = rot3x3;
        relXform.setTrans(currBone->getInitialPosition());

        //Transformasi F(j)
        Matrix4 fullXform = currBone->_getFullTransform();

        //Transformasi inv(A(j))
```

```
        Matrix4 invBindXform = currBone->_getBindingPoseInverseTransform();

        //Membuat transformasi C(j)
        Matrix4 currXform;
        currXform = Matrix4::IDENTITY;
        Quaternion currQuat= currBone->getOrientation();
        currQuat.ToRotationMatrix(rot3x3);
        currXform = rot3x3;
        Vector3 currPos=currBone->getPosition();
        currXform.setTrans(currPos);

        //Membuat transformasi inv(R(j))
        Matrix4 invMat =  relXform.inverse();

        //Menghitung nilai T(j)
        Matrix4 deltaXform =  invMat*currXform;

        //Mengambil quaternion dari T(j) untuk SLERP
        Quaternion deltaQuat = deltaXform.extractQuaternion();

        //Mengambil translasi dari T(j) untuk membuat T(j) baru
        Vector3 deltaTrans = deltaXform.getTrans();

        mFullMatrix[j] = fullXform;
        mInitialMatrix[j] = relXform;
        mCurrQuaternion[j] = deltaQuat;
        mCurrTrans[j]= deltaTrans;
        mBindPoseInverseMatrix[j] = invBindXform;
        mInitialMatrix[j] = relXform;
        mLocalMatrix[j]=currXform;

}//End loop bone

//-------------------
//Perhitungan vertex
//-------------------
for(unsigned int k=0; k<mVertexCount; k++)
{
        unsigned char Bone1, Bone2; //bone j dan parent j
        Real TInterp; //interpolan t

        Bone1 = boneArray[k*4];
        Bone2 = boneArray [1+(k*4)];

        TInterp = tArray[k];

        Quaternion mBlendedQuat;
        if (TInterp==0.0)
        {
                mBlendedQuat = mCurrQuaternion[(int)Bone1];
        }
        else if (TInterp==1.0)
        {
                mBlendedQuat = mCurrQuaternion[(int)Bone2];
```

```
        }
        else
        {
                mBlendedQuat =
Quaternion::Slerp (TInterp, mCurrQuaternion[Bone1], mCurrQuaternion[Bone2]);
        }

        //Membat matriks T(j) baru
        Matrix4 mCurrMatrix;
        Matrix3 rot3x3

        mCurrMatrix = Matrix4::IDENTITY;
        mBlendedQuat.ToRotationMatrix(rot3x3);

        mCurrMatrix = rot3x3;
        Vector3 mTrans = (mCurrTrans[Bone1]);
        mCurrMatrix.setTrans(mTrans);

        Matrix4 mBlendedMatrix=
                mInitialMatrix[(int)Bone1]*mCurrMatrix; //C(j)=R(j).Tnew(j)
        if (Bone1>0) mBlendedMatrix =
                mFullMatrix[Bone1-1] * mBlendedMatrix;//if bone1>0,F(j)= F(parent j).C(j)
        mBlendedMatrix =
                mBlendedMatrix*mBindPoseInverseMatrix[Bone1]; //M(j)=F(j).inv(A(j))

        //Blended vertex pos
        mDestPos[k].x = (mBlendedMatrix[0][0] * mVertices[k].x +
                         mBlendedMatrix[0][1] * mVertices[k].y +
                        mBlendedMatrix[0][2] * mVertices[k].z +
                        mBlendedMatrix[0][3]);

        mDestPos[k].y = (mBlendedMatrix[1][0] * mVertices[k].x +
                          mBlendedMatrix[1][1] * mVertices[k].y +
                          mBlendedMatrix[1][2] * mVertices[k].z +
                          mBlendedMatrix[1][3]);

        mDestPos[k].z = (mBlendedMatrix[2][0] * mVertices[k].x +
                         mBlendedMatrix[2][1] * mVertices[k].y +
                         mBlendedMatrix[2][2] * mVertices[k].z +
                         mBlendedMatrix[2][3]);

        //Blended normal

        mDestNorm[k].x = (mBlendedMatrix[0][0] * mNormals[k].x +
                          mBlendedMatrix[0][1] * mNormals[k].y +
                          mBlendedMatrix[0][2] * mNormals[k].z );

        mDestNorm[k].y = (mBlendedMatrix[1][0] * mNormals[k].x +
                          mBlendedMatrix[1][1] * mNormals[k].y +
                          mBlendedMatrix[1][2] * mNormals[k].z );

        mDestNorm[k].z = (mBlendedMatrix[2][0] * mNormals[k].x +
                          mBlendedMatrix[2][1] * mNormals[k].y +
                          mBlendedMatrix[2][2] * mNormals[k].z );
}//end looping vertex
```

```
//free mem
delete[] mFullMatrix;
delete[] mInitialMatrix;
delete[] mCurrQuaternion;
delete[] mCurrTrans;
delete[] mBindPoseInverseMatrix;
delete[] mLocalMatrix;
```

4.3.10. Implementasi Subproses Output Render Animasi

Pada subproses ini dilakukan perintah-perintah untuk menampilkan data-data yang telah berubah karena perubahan matriks proyeksi maupun perubahan *skeletal animation*. Setelah menampilkan *output* yang berupa tampilan tiga dimensi, subproses ini akan melakukan proses *idle* dimana akan dilakukan pembacaan tombol yang ditekan maupun gerakan tetikus. Subproses ini sepenuhnya dijalankan secara automatis oleh *API*. Di dalam subproses inilah dilakukan perintah-perintah baik dari DirectX maupun OpenGL untuk me*render* tampilan.

Karena subproses ini sepenuhnya dilakukan oleh *API*, maka *pseudo-code* dari subproses ini tidak dapat ditentukan.

4.3.11. Impementasi Subproses Input Animasi, Mode, Kamera, dan Keluar

Di dalam subproses ini objek *FrameListener* dijalankan. Di dalam objek ini, sudah ditentukan beberapa kondisi penekanan tombol dan gerakan tetikus sebagai berikut:

- *Esc*: Keluar dari aplikasi
- *Tombol Panah pada keyboard*: Translasi kamera
- *Gerakan tetikus*: Rotasi kamera

- *Tombol 1 s/d 7*: Pemindahan kamera ke beberapa posisi yang telah ditentukan dengan posisi kamera nomor 1 sebagai *default*.
- *Tombol Q/W/E*: Pemilihan mode *skin deformation*. *Q* untuk memilih *basic skeletal animation*, *W* untuk memilih *vertex blending*, dan *E* untuk memilih *bones blending*.
- *Tombol Z/X*: Pemilihan pengaktifan rotasi pada model. *Z* untuk memilih rotasi berdasarkan sumbu Y, dan *X* untuk memilih rotasi berdasarkan sumbu Z.

Apabila *FrameListener* mengenali salah satu masukan di atas, maka subproses akan kembali ke subproses Input Animasi, Mode dan Kamera pada prose Inti. Kecuali kalau *FrameListener* mengenali bahwa tombol *esc* ditekan. Maka subproses akan menjalankan proses de-inisialisasi dan akan keluar ke sistem utama.

Dalam subproses ini terdapat dua (2) macam inputan yaitu dari keyboard dan tetikus. Sehingga metode untuk mengenali inputan ini juga ada dua yaitu "KeyPressed(KeyEvent* e)" untuk mengenali masukan dari keyboard, dan "MouseMoved(MouseEvent* e)" untuk mengenali masukan dari tetikus (yang berupa gerakan tetikus).

Pseudo-code dari subproses ini adalah:

```
  virtual void keyPressed(KeyEvent* e)
  {
//Switch mendeteksi key
    switch ( e->getKey( ) )
    {
//Key Escape = keluar
          case KC_ESCAPE:
       mContinue = false;
         delete[] mVertices;
         delete[] mNormals;
```

```
            delete[] mBoneIndices;
            delete[] mTWeights;

        break;
//Kamera 1 s/d 7
    case KC_1:
        mCameraMode=1;
        mCamera->getParentSceneNode()->detachObject( mCamera );
        mCamNode = mSceneMgr->getSceneNode( "CamNode1" );
        mSceneMgr->getSceneNode( "PitchNode1" )->attachObject( mCamera );
        break;

    case KC_2:
        mCameraMode=2;
        mCamera->getParentSceneNode()->detachObject( mCamera );
        mCamNode = mSceneMgr->getSceneNode( "CamNode2" );
        mSceneMgr->getSceneNode( "PitchNode2" )->attachObject( mCamera );
        break;

    case KC_3:
        mCameraMode=3;
        mCamera->getParentSceneNode()->detachObject( mCamera );
        mCamNode = mSceneMgr->getSceneNode( "CamNode3" );
        mSceneMgr->getSceneNode( "PitchNode3" )->attachObject( mCamera );
        break;

    case KC_4:
        mCameraMode=4;
        mCamera->getParentSceneNode()->detachObject( mCamera );
        mCamNode = mSceneMgr->getSceneNode( "CamNode4" );
        mSceneMgr->getSceneNode( "PitchNode4" )->attachObject( mCamera );
        break;

    case KC_5:
        mCameraMode=5;
        mCamera->getParentSceneNode()->detachObject( mCamera );
        mCamNode = mSceneMgr->getSceneNode( "CamNode5" );
        mSceneMgr->getSceneNode( "PitchNode5" )->attachObject( mCamera );
        break;

    case KC_6:
        mCameraMode=6;
        mCamera->getParentSceneNode()->detachObject( mCamera );
        mCamNode = mSceneMgr->getSceneNode( "CamNode6" );
        mSceneMgr->getSceneNode( "PitchNode6" )->attachObject( mCamera );
        break;


    case KC_7:
        mCameraMode=7;
        mCamera->getParentSceneNode()->detachObject( mCamera );
        mCamNode = mSceneMgr->getSceneNode( "CamNode7" );
        mSceneMgr->getSceneNode( "PitchNode7" )->attachObject( mCamera );
        break;
```

```
//Gerakan dengan panah+pUp pDown
    case KC_UP:
      mDirection.z -= mMove;
                //mDirection.z = -mMove;
      break;

    case KC_DOWN:
      mDirection.z += mMove;
                //mDirection.z = mMove;
      break;

    case KC_LEFT:
      mDirection.x -= mMove;
                //mDirection.x = -mMove;
      break;

    case KC_RIGHT:
      mDirection.x += mMove;
                //mDirection.x = mMove;
      break;

    case KC_PGDOWN:
      mDirection.y -= mMove;
      break;

    case KC_PGUP:
      mDirection.y += mMove;
      break;

//Tombol toggle animasi 1 / 2
    case KC_Z:
        if (!Animasi)
        {
            mSumbuAnimasi=1;
            Animasi=true;
            mTotalSumbuAnimasi=0;
        }
        break;

    case KC_X:
        if (!Animasi)
        {
            mSumbuAnimasi=2;
            Animasi=true;
            mTotalSumbuAnimasi=0;
        }
        break;

//Tombol toggle blending mode
  case KC_Q:
        Mode=BSA;
        mSceneMgr->getSceneNode( "ModelNode" )->detachAllObjects();
        mSceneMgr->getSceneNode( "ModelNode" )
              ->attachObject(mSceneMgr->getEntity("TASimpleBSA"));
```

```
            break;

    case KC_W:
            Mode=VB;
            mSceneMgr->getSceneNode( "ModelNode" )->detachAllObjects();
            mSceneMgr->getSceneNode( "ModelNode" )
                  ->attachObject(mSceneMgr->getEntity("TASimpleVB"));
            break;

    case KC_E:
            Mode=BB;
            mSceneMgr->getSceneNode( "ModelNode" )->detachAllObjects();
            mSceneMgr->getSceneNode( "ModelNode" )
                  attachObject(mSceneMgr->getEntity("TASimpleBB"));
            break;
}
void mouseMoved(MouseEvent* e)
  {
     mCamNode->yaw( Degree(-e->getRelX( ) * mRotate) );
     mCamNode->getChild( 0 )->pitch( Degree(-e->getRelY() * mRotate) );
  }
```

4.3.12. Implementasi Subproses De-inisialisasi Antarmuka

Subproses ini akan melakukan de-inisialisasi antarmuka yang berupa *FrameListener*. Subproses ini dilakukan secara automatis oleh *API* ketika aplikasi dihentikan dengan menyatakan *mContinue=false*;.

Karena subproses sepenuhnya ini dilakukan oleh *API*, maka *pseudo-code* dari subproses ini tidak dapat ditentukan.

4.3.13. Implementasi Subproses De-inisialisasi Data dan Objek Bergerak

Subproses ini akan melakukan de-inisialisasi data-data yang berupa *Mesh, Entity, Texture,* dan lain-lain. Subproses ini dilakukan secara automatis oleh *API* ketika aplikasi dihentikan dengan menyatakan *mContinue=false*;.

Karena subproses sepenuhnya ini dilakukan oleh *API*, maka *pseudo-code* dari subproses ini tidak dapat ditentukan.

4.3.14. Implementasi Subproses De-inisialisasi Lingkungan Grafika

Subproses ini akan melakukan de-inisialisasi lingkungan grafika yang berupa *Root, SceneManager, ResourceManager, RenderSystem,* dan lain-lain. Subproses ini dilakukan secara automatis oleh *API* ketika aplikasi dihentikan dengan menyatakan *mContinue=false;*.

Karena subproses sepenuhnya ini dilakukan oleh *API*, maka *pseudo-code* dari subproses ini tidak dapat ditentukan.

4.4. Implementasi Antarmuka

Seperti yang sudah dijelaskan sebelumnya, antarmuka dari aplikasi ini hanya berupa masukan dari keyboard dan tetikus antara lain:

- Pemilihan keluar dari aplikasi dengan menekan tombol *esc*.
- Pemilihan penempatan kamera pada *node-node* yang telah ditentukan sebelumnya dengan menekan salah satu tombol *1/2/3/4/5/6/7*.
- Translasi dan rotasi kamera dengan tombol panah dan gerakan tetikus.
- Pemilihan mode *skin deformation* dengan menekan salah satu tombol *Q/W/E*.
- Pengaktifan animasi dengan menekan salah satu tombol *Z/X*.

Aplikasi ini tidak menggunakan *button* atau *form* seperti dalam *MFC (Microsoft Foundation Class)* karena jika menggunakan MFC, maka automatis aplikasi akan berjalan pada mode *windows* dan akan mengurangi performa

aplikasi. Sehingga sedikit-banyak hal ini akan menyebabkan konsep *real time* dari aplikasi ini menjadi berkurang.

Hal ini pula yang menjadi bahan pertimbangan untuk mengimplementasikan aplikasi ini dalam mode *fullscreen* ketika dijalankan.

BAB V
UJI COBA DAN ANALISIS HASIL

BAB V
UJI COBA DAN ANALISIS HASIL

Perangkat lunak yang dibangun pada Tugas Akhir ini dirancang menampilkan beberapa kondisi implementasi *real time skin deformation* dengan menggunakan metode *basic skeletal animation/single weighted skinning*, *vertex blending/multi weighted skinning*, dan *bones blending* serta membandingkan antara ketiga metode tersebut. Pada bab ini akan dijelaskan uji coba yang telah dilakukan terhadap perangkat lunak serta evaluasi berdasarkan hasil uji coba tersebut.

5.1. Lingkungan dan pelaksanaan Uji Coba

Uji coba dilakukan pada komputer dengan spesifikasi perangkat keras sebagai berikut :

- Prosesor AMD Duron 1,2 GHz
- Memori 512 MB
- Graphics card ATi Radeon 9550 GT
- Harddisk 120 GB
- Sistem operasi Microsoft XP Service Pack 1
- API OGRE 3D versi 1.0.6

Ada beberapa uji coba dengan tujuan berbeda yang dilakukan, yaitu :

- Uji Coba I : untuk menguji kebenaran dari tampilan metode *basic skeletal animation, vertex blending,* dan *bones blending* pada sebuah model dengan jumlah polygon sedikit (low polygon model)
- Uji Coba II : untuk menguji kebenaran dari tampilan metode *basic skeletal animation, vertex blending,* dan *bones blending* pada sebuah model dengan jumlah polygon banyak (high polygon model)

Sedangkan untuk masing-masing uji coba, dilaksanakan sub uji coba untuk membuktikan kebenaran perubahan rotasi dari suatu tulang terhadap masing-masing model yang besarnya:

- 0° atau kondisi awal model
- Rotasi 90° terhadap sumbu –Z (negatif Z)
- Rotasi 90° terhadap sumbu X
- Rotasi 180° terhadap sumbu X

Rotasi pada sumbu Y tidak dipakai untuk pembuktian pada Tugas Akhir ini karena hasilnya akan sama saja dengan rotasi pada sumbu -Z. Dan rotasi sebesar 180° terhadap sumbu -Z tidak dipakai karena merupakan suatu kondisi yang abnormal dimana *mesh* akan bertemu satu sama lainnya.

5.2. Pelaksanaan Uji Coba dengan Low-Polygon Model

Pada subbab ini akan dijelaskan mengenai uji coba yang dilakukan untuk membuktikan kebenaran dari metode *basic skeletal animation, vertex blending,*

dan *bones blending* jika menggunakan sebuah model berpoligon sedikit (low-polygon model).

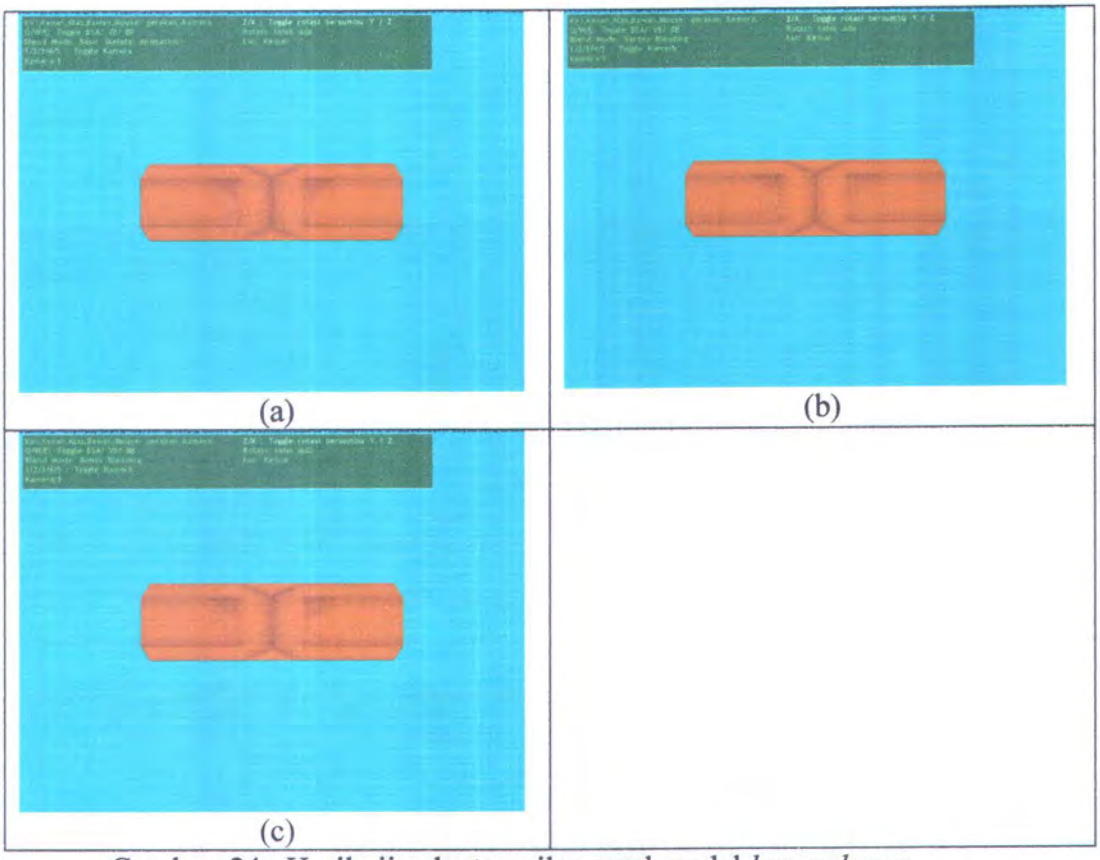
Berikut ini adalah data-data tentang model *low-polygon* yang dipakai:

Jumlah Vertex :	78
Jumlah Polygon :	152
Jumlah Tulang :	4
Dimensi:	
X:	160,158 cm
Y:	50,178 cm
Z:	40,12 cm

Tabel 1. Tabel properti dari model *low-polygon*

5.2.1. Uji Coba tampilan awal model

Uji coba pertama dilakukan untuk melihat kebenaran tampilan metode *basic skeletal animation, vertex blending, dan bones blending* pada posisi awal (ketika belum terjadi rotasi) dari model *low-polygon.*



Gambar 24. Hasil uji coba tampilan awal model *low-polygon*
(a) Basic Skeletal Animation
(b) Vertex Blending
(c) Bones Blending

Dari hasil uji coba tersebut, dapat dilihat bahwa tampilan awal *basic skeletal animation, vertex blending,* dan *bones blending* adalah sama. Hal ini membuktikan kebenaran bahwa persamaan dalam metode-metode tersebut tidak mengubah posisi verteks sebelum ada transformasi.

5.2.2. Uji Coba tampilan rotasi 90 derajat terhadap sumbu -Z.

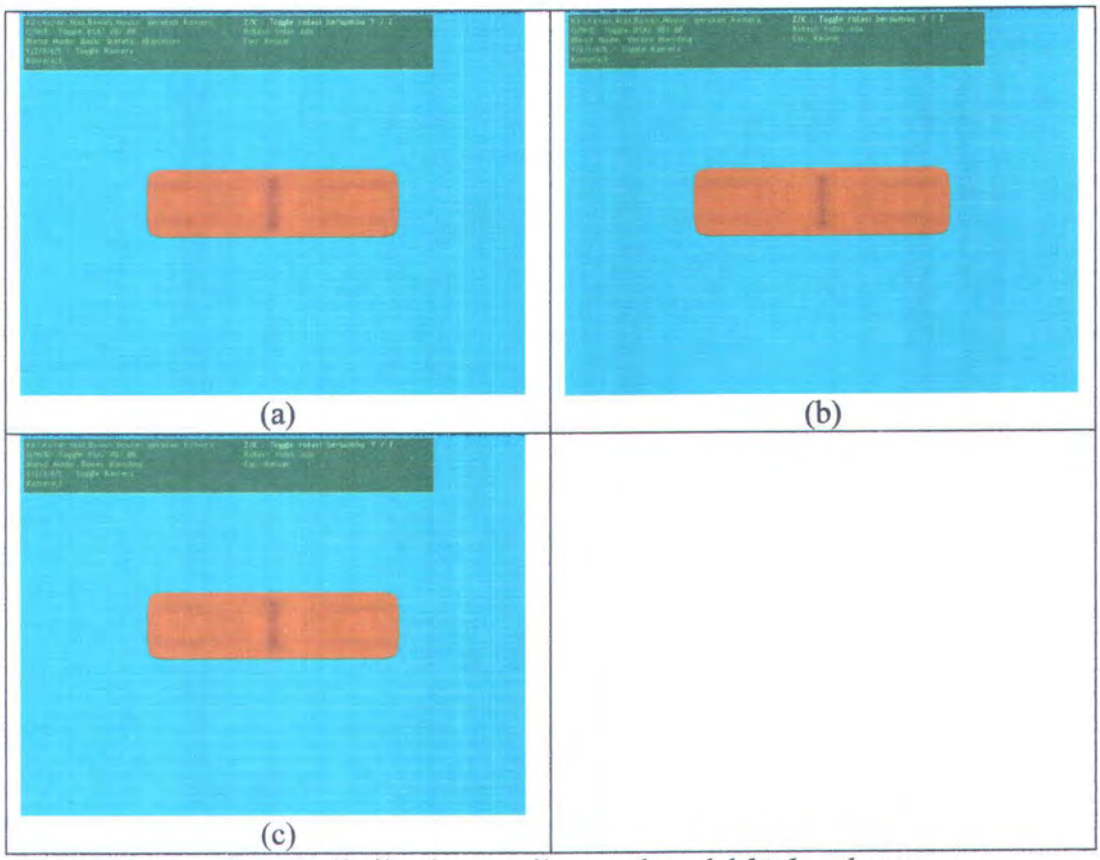
Uji coba kedua ini dilakukan untuk melihat kebenaran tampilan metode *basic skeletal animation, vertex blending, dan bones blending* pada model *low-polygon* ketika terjadi rotasi sebesar 90 derajat terhadap sumbu -Z.

Jumlah Vertex :	1250
Jumlah Polygon :	2496
Jumlah Tulang :	4
Dimensi:	
X:	160,158 cm
Y:	43,744 cm
Z:	40,12 cm

Tabel 2. Tabel properti dari model *high-polygon*

5.3.1. Uji Coba tampilan awal model

Uji coba pertama dilakukan untuk melihat kebenaran tampilan metode *basic skeletal animation, vertex blending, dan bones blending* pada posisi awal (ketika belum terjadi rotasi) dari model *high-polygon.*

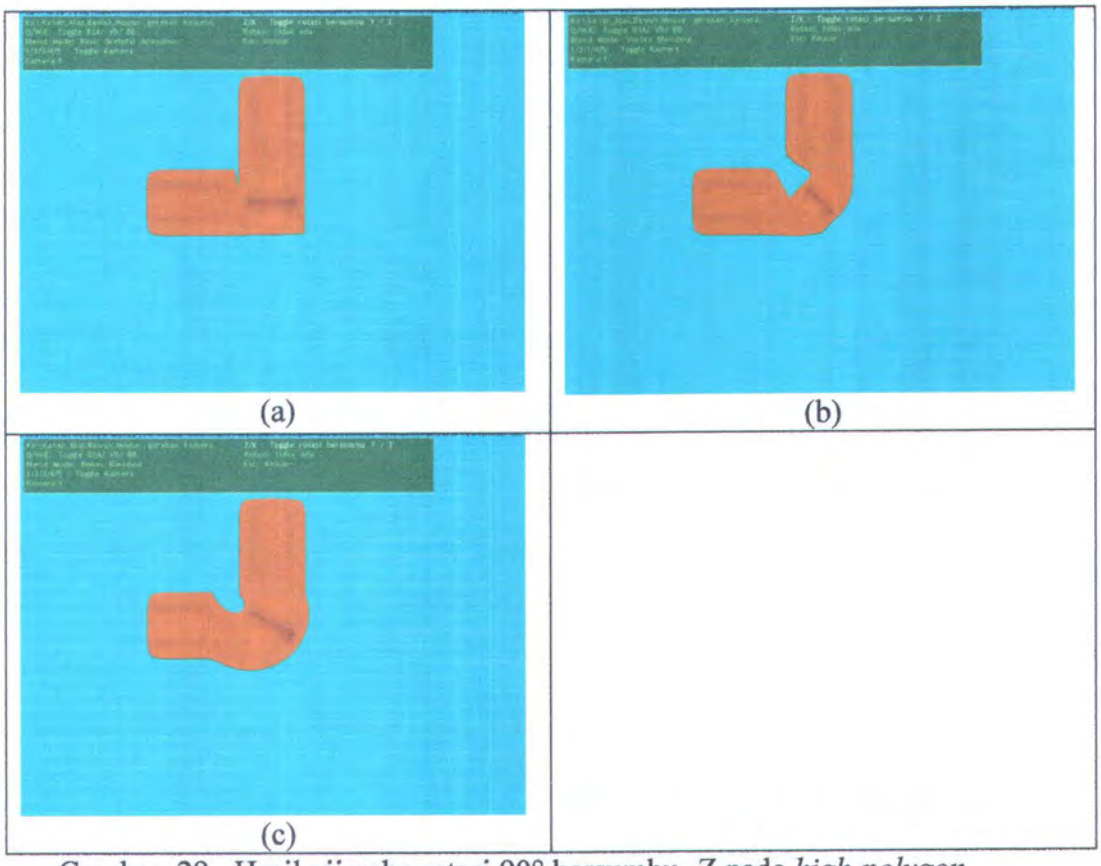


Gambar 28. Hasil uji coba tampilan awal model *high-polygon*
(a) Basic Skeletal Animation
(b) Vertex Blending
(c) Bones Blending

Dari hasil uji coba tersebut, dapat dilihat bahwa tampilan awal *basic skeletal animation, vertex blending,* dan *bones blending* adalah sama. Hal ini membuktikan kebenaran bahwa persamaan dalam metode-metode tersebut tidak mengubah posisi verteks sebelum ada transformasi.

5.3.2. Uji Coba tampilan rotasi 90 derajat terhadap sumbu -Z.

Uji coba kedua ini dilakukan untuk melihat kebenaran tampilan metode *basic skeletal animation, vertex blending, dan bones blending* pada model *high-polygon* ketika terjadi rotasi sebesar 90 derajat terhadap sumbu -Z.



Gambar 29. Hasil uji coba rotasi 90° bersumbu -Z pada *high-polygon*
(a) Basic Skeletal Animation
(b) Vertex Blending
(c) Bones Blending

Dari hasil uji coba tersebut dapat dilihat bahwa pada basic skeletal animation, perubahan posisi verteks menyebabkan kulit menjadi tidak halus. Mesh menjadi kelihatan kaku seperti sebuah besi yang dipatahkan. Sedangkan pada vertex blending dan bones blending, tampilan menjadi lebih halus. Pada bones blending tampilan terlihat lebih bulat dari vertex blending. Tampilan pada model ini tampak lebih halus dari ketika menggunakan model low-polygon karena jumlah vertex pada model ini lebih banyak sehingga lebih banyak pula verteks yang terdeformasi.

5.3.3. Uji Coba tampilan rotasi 90 derajat terhadap sumbu X.

Uji coba ketiga ini dilakukan untuk melihat kebenaran tampilan metode *basic skeletal animation, vertex blending, dan bones blending* pada model *high-polygon* ketika terjadi rotasi sebesar 90 derajat terhadap sumbu X.

sumbu X. Posisi verteks sekarang menjadi kebalikan dari posisi verteks terdahulu. Sehingga *mesh* yang terdeformasi semakin tertarik melewati tengah-tengah model. Tampilan masih belum halus. Pada tampilan *vertex blending* masih terdapat artifak *"twisting elbow"* dimana perataan posisi rata-rata verteks menyebabkan verteks yang terdeformasi berada tepat di tengah-tengah model. Hal ini menyebabkan suatu tampilan abnormal yang membuat *mesh* menjadi sempit dan tampak seperti menghilang. Hal ini terjadi karena hasil perataan posisi verteks adalah tepat di tengah-tengah model, sehingga terjadi penumpukan verteks di tengah-tengah model tersebut. Sedangkan pada *bones blending* tidak terdapat artifak *"twisting elbow"*. Jarak verteks terdeformasi terhadap pusat model juga relatif sama dengan sebelumnya sehingga diameter penampang relatif sama. *Mesh* hanya terpelintir saja tanpa tertarik masuk ke dalam model. Hal ini karena perataaan dilakukan pada sudut rotasi, bukan pada posisi akhir verteks.

5.4. Perbandingan Metode

Perbandingan keunggulan dan kelemahan dari metode-metode *skin deformation* yaitu *basic skeletal animation*, *vertex blending*, dan *bones blending* adalah sebagai berikut:

No	Item	BSA	VB	BB
1	Model yang dipakai	*Single-weighted* (satu tulang per verteks)	*Multi-weighted* (banyak tulang per verteks)	*Single-weighted* (satu tulang per verteks). Hanya bisa dipakai pada model dengan tulang-tulang yang mempunyai *child* maksimal satu.

2	Tampilan deformasi	Kasar. Terdapat artifak *"twisting elbow"* ketika terpelintir 180°	Halus. Terdapat artifak *"twisting elbow"* ketika terpelintir 180°	Halus. Tidak terdapat artifak *"twisting elbow"* ketika terpelintir 180°
3	Banyak instruksi sebelum *rendering*	Loop per tulang yang di dalamnya terdapat: - *n* buah perkalian antar matriks (64 buah perkalian & 64 buah penambahan) per *n* tulang *parent* dari *root* ke tulang secara hierarkikal.	Loop per tulang yang di dalamnya terdapat: - *n* buah perkalian antar matriks (64 buah perkalian & 64 buah penambahan) per *n* tulang *parent* dari *root* ke tulang secara hierarkikal.	• Loop per tulang yang di dalamnya terdapat: - *n* buah perkalian antar matriks (64 buah perkalian & 64 buah penambahan) per *n* tulang *parent* dari *root* ke tulang secara hierarkikal. • Loop per verteks yang di dalamnya terdapat: - Sebuah perkalian matriks dengan vektor (16 buah perkalian & 16 buah penambahan) - Sebuah perkalian titik (*dot product*) (4 buah perkalian & 4 buah penambahan) - Sebuah operasi perataan (2 buah pengurangan & sebuah pembagian)
4	Banyak instruksi pada saat *rendering*	• Loop per tulang yang di dalamnya terdapat: - *n*X2 buah perkalian	• Loop per tulang yang di dalamnya terdapat: - *n*X2 buah perkalian	• Loop per tulang yang di dalamnya terdapat: - *n*X2 buah perkalian antar matriks (128

		antar matriks (128 buah perkalian & 128 buah penambahan) per *n* tulang *parent* dari *root* ke tulang secara hierarkikal. • Loop per verteks yang di dalamnya terdapat: - Perkalian matriks dengan vektor (16 buah perkalian & 16 buah penambahan)	antar matriks (128 buah perkalian & 128 buah penambahan) per *n* tulang *parent* dari *root* ke tulang secara hierarkikal. • Loop per verteks yang di dalamnya terdapat: - Loop per tulang pasangan yang di dalamnya terdapat *n* buah perkalian matriks dengan vektor (16 buah perkalian & 16 buah penambahan) per *n* tulang pasangan. - *n* buah perkalian skalar dengan vektor (4 buah perkalian) per *n* tulang pasangan. - *n* buah penambahan vektor (4 buah penambahan) per *n* tulang pasangan.	buah perkalian & 128 buah penambahan) per *n* tulang *parent* dari *root* ke tulang secara hierarkikal. • Loop per verteks yang di dalamnya terdapat: - Operasi *SLERP* yang terdiri dari: ▪ Sebuah percabangan *if / else* ▪ Sebuah perkalian titik (*dot product*) (4 buah perkalian & 4 buah penambahan) ▪ Tiga buah operasi sinus ▪ Dua buah perkalian skalar ▪ Sebuah pengurangan skalar ▪ Sebuah pembagian skalar ▪ Dua buah perkalian vektor dengan skalar (8 buah perkalian & 8 buah penambahan) ▪ Sebuah penambahan antar vektor

				(4 buah penambahan) ▪ Negasi vektor (4 buah perkalian dengan nilai -1) jika sudut ≥ 180° ▪ Operasi normalisasi (4 buah perkalian, 4 buah penambahan, sebuah operasi akar kuadrat, & 4 buah pembagian) jika sudut ≥ 180° - Sebuah perkalian antar matriks (64 buah perkalian & 64 buah penambahan) - Perkalian matriks dengan vektor (16 buah perkalian & 16 buah penambahan)

Tabel 3. Tabel perbandingan metode skin deformation

Pada tabel di atas, dapat dilihat perbandingan kelebihan dan kekurangan masing-masing metode. Pada *basic skeletal animation*, tampilan masih tampak kasar dan kulit tampak seperti kaku serta masih tampak artifak *"twisting elbow"*. Tetapi metode *basic skeletal animation* ini adalah metode yang paling sederhana sehingga proses *rendering* akan berjalan. lebih cepat. Pada *vertex blending*,

tampilan sudah tampak halus, tetapi masih terdapat artifak *"twisting elbow"*. Metode *vertex blending* ini mempunyai sedikit lebih banyak instruksi sehingga lebih lambat sedikit lambat daripada *basic skeletal animation.* Sedangkan pada *bones blending*, tampilan sudah tampak halus dan tidak ada artifak *"twisting elbow"*. Tetapi, metode ini mempunyai instruksi lebih banyak daripada *basic skeletal animation* dan *vertex blending*, sehingga proses *rendering* akan berjalan lebih lambat daripada *basic skeletal animation* maupun *vertex blending*. Selain itu, metode ini tidak dapat diterapkan pada suatu model yang tulang-tulangnya mempunyai *child* lebih dari satu (seperti tulang telapak tangan).

BAB VI
KESIMPULAN DAN SARAN

BAB VI
KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan yang dapat diambil berdasarkan uji coba yang telah dilakukan. Selanjutnya, diberikan beberapa saran yang mungkin dapat mengembangkan hasil yang telah diperoleh pada tugas akhir ini.

6.1. Kesimpulan

Kesimpulan yang dapat diambil dari uji coba yang telah dilakukan adalah :

1. Pada *basic skeletal animation*, tampilan masih belum halus karena tidak adanya perataan posisi verteks untuk memperhalus *mesh*.
2. Pada *vertex blending*, tampilan menjadi lebih halus karena dilakukan perataan posisi verteks berdasarkan posisi verteks jika dimplementasikan matriks transformasi masing-masing tulang pasangan.
3. Apabila salah satu hasil transformasi tulang pasangan verteks merupakan perputaran sebesar 180 derajat, maka hasil perataan akan berada di tengah-tengah model sehingga menyebabkan suatu artifak yang bernama *"twisting elbow"* atau *"candy wrapper"* yang menyebabkan *mesh* tampak seperti menghilang.
4. Pada *bones blending* tampilan sudah halus karena terjadi perataan pada sudut rotasi.
5. Pada *bones blending* artifak *"twisting elbow"* atau *"candy wrapper"* tidak tampak.

6. Terdapat beberapa kelemahan dari metode *bones blending*. Salah satunya adalah diperlukan suatu proses ekstra untuk menentukan interpolan yang akan digunakan dalam perataan. Walaupun proses ini hanya akan dijalankan sekali saja dan nilainya tidak akan berubah, tetap saja akan mempengaruhi kinerja terutama di awal-awal aplikasi.
7. Selain itu juga metode *bones blending* hanya dapat berlaku pada tulang yang hanya mempunyai satu tulang *child* saja. Apabila tulang pasangan verteks mempunyai lebih dari satu tulang *child* (seperti tulang telapak tangan), maka metode ini tidak akan dapat digunakan karena untuk mencari interpolan hanya dapat dilakukan pada dua buah tulang saja. Metode perataan sudut rotasi dengan menggunakan *SLERP* juga hanya bisa dilakukan dengan input dua buah *quaternion* rotasi saja.
8. Walaupun mempunyai tampilan yang lebih baik, tetapi metode *bones blending* ini kurang efektif digunakan, terutama pada model yang salah satu tulangnya mempunyai *child* lebih dari satu.

6.2. Saran

Saran-saran yang dapat diberikan untuk pengembangan tugas akhir ini adalah :

1. Mengembangkan aplikasi dengan menggunakan metode *Spherical Blend Skinning* yang merupakan pengembangan dari *bones blending*.
2. Mengembangkan aplikasi dengan menggunakan metode *skin deformation* lain seperti *bone links* [07].

3. Mengembangkan uji coba agar melakukan uji coba *skin deformation* dengan menggunakan hardware (GPU). Bahasa pemrograman yang dipakai bisa berupa Cg, HLSL, atau GLSL.

DAFTAR PUSTAKA

[10] *Matrix and Quaternion FAQ,* http://skal.planet-d.net/demo/matrixfaq.htm .

[11] Petkovsek, C. *Quaternion and Rotation Primer*. OGRE Wiki, http://www.ogre3d.org/wiki/index.php/Quaternion_and_Rotation_Primer, **June 16th, 2005.**

[12] Watt,A. *3D Computer Graphics Third Edition.* Pearson Education Limited, 2000.

[13] Hearn, D., Baker, M.P. *Computer Graphics C Version Second Edition.* Prentice Hall, 1997.

[14] Eberly, D. *A Linear Algebraic Approach to Quaternions.* **Geometric Tools, Inc.,** http://www.geometrictools.com , **September** 16th, 2002.

[15] Eberly, D. *Quaternion Algebra and Calculus*. Geometric Tools, **Inc.,** http://www.geometrictools.com , **September 27th, 2002.**

[16] Eberly, D. *Rotation Representations and Performance Issues*. **Geometric Tools, Inc.,** http://www.geometrictools.com , **September** 3rd, 2002.

[17] Eberly, D. *Euler Angle Formulas*. Geometric Tools, Inc., http://www.geometrictools.com , **December 1st, 1999.**