PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK ANIMASI KINEMATIK UNTUK PERGERAKAN MANUSIA

Oleh :

CAROLINE INNES DAMAYANTI
NRP. 2693 100 004

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
1999

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK ANIMASI KINEMATIK UNTUK PERGERAKAN MANUSIA

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
S u r a b a y a

Mengetahui / Menyetujui

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Ir. ARIF DJUNAIDY, M.Sc.	RULLY SOELAIMAN, S. Kom
NIP. 131 633 403	NIP. 132 085 802

S U R A B A Y A
Pebruari, 1999

Untuk *Ibu dan Almarhum Bapak* Tiada Kata Maupun Benda Apapun di Dunia Ini Yang Mampu Mengungkapkan Rasa Terima Kasihku

dan

Untuk *Roy Setyaeka Kurnia* Terimakasih Atas Kebahagiaan Yang Kudapatkan Selama Kita Bersama

ABSTRAK

Salah satu jenis penelitian di bidang grafika komputer adalah cara mengontrol dan memvisualisasikan gerakan manusia dalam bentuk animasi. Hal ini disebabkan oleh keinginan untuk menjadikan manusia sebagai aktor tiruan yang dapat diarahkan agar bergerak sesuai dengan perintah. Disini manusia dianggap sebagai sebuah figur yang mempunyai struktur artikulasi, yaitu sebuah struktur yang terdiri dari sejumlah link yang dihubungkan oleh joint.

Hasil dari tugas akhir ini berupa editor dengan fasilitas untuk menggerakkan bagian manusia hasil interaksi dengan pengguna. Untuk merepresentasikan figur manusia digunakan notasi yang sering dipakai dalam manipulasi robot yaitu Notasi Denavit-Hartenberg (D-H). Teknik yang digunakan untuk menentukan animasi pergerakan adalah kinematika invers. Langkah yang dilakukan adalah menentukan posisi dari salah satu *end-effector* dari figur manusia tersebut dan selanjutnya metode kinematika invers digunakan untuk melakukan perhitungan terhadap posisi link dan joint yang berhubungan dengan *end-effector* tadi. Proses dilanjutkan dengan melakukan penggambaran kerangka pada frame menggunakan fungsi OpenGL. Semua gambar frame yang dihasilkan divisualisasikan dalam bentuk animasi. Dari hasil perhitungan, besarnya kecepatan perhitungan kinematika invers tergantung pada besarnya jarak antara posisi *end-effector* sebelumnya dan posisi tujuan.

Tugas akhir ini bermanfaat untuk menampilkan figur manusia dalam bentuk animasi sehingga dapat digunakan sebagai simulasi pergerakan untuk industri film maupun periklanan. Dan untuk pengembangan selanjutnya jenis obyek yang dianimasikan dapat dibuat lebih fleksibel dan tidak terbatas pada manusia saja.

KATA PENGANTAR

Puji syukur kepada Allah Bapa di Surga, karena akhirnya penulis dapat menyelesaikan pembuatan Tugas Akhir yang berjudul :

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK ANIMASI KINEMATIK UNTUK PERGERAKAN MANUSIA

Tugas Akhir ini dikerjakan guna memenuhi persyaratan akademis dalam rangka ujian akhir bagi mahasiswa Strata 1 (S1) Jurusan Teknik Informatika, Institut Teknologi Sepuluh Nopember.

Bersama ini penulis mengucapkan terima kasih dan penghargaan yang sebesar-besarnya kepada pihak-pihak yang telah memberikan bantuan baik dalam material dan non-material, antara lain :

1. DR Ir Arif Djunaidy , selaku Ketua Jurusan Teknik Informatika FTI-ITS sekaligus Dosen Pembimbing I yang telah banyak membimbing penulis dalam penyelesaian tugas akhir ini.
2. Rully Soelaiman, S. Kom selaku Dosen Pembimbing II yang telah banyak membantu dan membimbing penulis dalam penyelesaian tugas akhir ini.
3. DR Ir Supeno Djanali selaku Dosen Wali penulis semasa kuliah.
4. Yudhi Purwananto, S. Kom yang telah banyak membantu memberikan masukan, bimbingan dan tanggapan.
5. Kakak-kakakku Mas Ferdy, Mbak Mem, dan si Rufi yang memberikan dorongan moral dan material.
6. Argo Among Negoro, S. Kom atas dorongan moril dan bantuan dalam mencari buku referensi yang sangat berguna.

7. Rekan Denni Retnani yang menjadi sahabat dikala suka dan duka, serta rekan Windy Agustina yang menjadi teman seperjuangan dan sepenanggungan bersama.
8. Rekan Gatot Sulistyo atas masukan, bantuan dan kebaikannya semasa kuliah dan di luar kuliah dan rekan Yoseph Denny atas penjualan komputernya yang tidak pernah rewel di waktu TA.
9. Rekan Widiono yang telah banyak membantu penulis semasa perkuliahan dan mengajarkan bagaimana cara membuat program yang baik.
10. Rekan Ary Nur yang menjadi teman baik dan secara tidak sadar sering memberikan hiburan dan bantuan.
11. Serta semua pihak lain, telah memberikan pengertian dan bantuan selama Tugas Akhir dan tidak mungkin disebutkan satu-persatu

Akhirnya penulis berharap agar Tugas Akhir ini dapat memberikan manfaat dan inspirasi kepada kita semua. Penulis menyadari bahwa masih terdapat kelemahan dalam penyusunan Tugas Akhir ini. Untuk itu kritik dan saran yang membangun sangat diharapkan guna menambah manfaat serta mengurangi kesalahan dan kekurangan yang ada.

Surabaya, Februari 1999

Caroline Innes D

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1. Latar Belakang

Perancangan animasi benda tiga dimensi sudah berkembang dalam tahap animasi terhadap obyek-obyek yang kompleks. Pembuatan program yang digunakan untuk mengontrol pergerakan animasi komputer telah muncul sebagai salah satu jenis penelitian dalam bidang grafika komputer. Salah satu bentuk obyek kompleks yang sudah dikenal adalah figur yang berartikulasi. Animasi terhadap figur yang berartikulasi dirasakan sudah menjadi populer belakangan ini. Hal ini disebabkan oleh keinginan untuk menjadikan figur tersebut sebagai aktor tiruan yang dapat diarahkan agar bergerak sesuai dengan perintah. Sebagai salah satu contoh adalah munculnya film animasi *Toys story* yang keseluruhannya menggunakan sarana komputer untuk membentuk gambar-gambar pada filmnya. Pembentukan figur dapat dilakukan dengan menciptakan pendekatan geometris terhadap obyek. Figur yang berartikulasi dapat dibentuk dengan menghubungkan bagian-bagian link yang ada dengan sebuah titik joint. Kumpulan dari link dan joint inilah yang nantinya akan menjadi sebuah figur yang pergerakannya dapat diatur dengan mengubah posisi-posisinya.

Pada dasarnya cara melakukan animasi terhadap figur ini adalah dengan melakukan kontrol terhadap kerangka figur yang ada dan kemudian dilanjutkan pada tahap pewarnaan terhadap kerangka tersebut. Permasalahannya adalah

bagaimana cara membuat animasi yang alami terhadap setiap pergerakan yang terjadi. Untuk itulah digunakan teknik kinematika invers. Yang perlu dilakukan hanyalah menentukan posisi dari sebuah joint, perubahan yang terjadi didapat dari perhitungan terhadap joint lain yang merupakan turunannya.

Motivasi dari tugas akhir ini adalah meningkatkan kemampuan animasi dengan penambahan interaksi yang aktif dari pengguna, yang akan digunakan untuk menciptakan pergerakan. Untuk ini proses yang dilibatkan lebih ditekankan pada cara-cara bagaimana melakukan animasi kinematik pada sebuah obyek, dan pengambilan manusia sebagai obyek pergerakan adalah berdasarkan kompleksitas hubungan yang sudah umum dan lebih mudah untuk menunjukkan hasil pergerakan alaminya.

1.2. Tujuan dan Manfaat

Tujuan dari tugas akhir ini adalah membuat perangkat lunak yang mampu menampilkan implementasi animasi dari sebuah obyek manusia sebagai salah satu bentuk yang mempunyai titik artikulasi dengan metode kinematika invers. Diharapkan perangkat lunak yang dihasilkan ini mampu menampilkan animasi terhadap manusia yang bergerak sehingga bermanfaat untuk industri perfilman maupun periklanan.

1.3. Permasalahan

Untuk dapat membuat perangkat lunak yang dapat menyediakan fasilitas editor yang mampu memberikan interaksi aktif dalam menganimasikan obyek manusia terdapat beberapa sub-problem , yaitu :

- Pembentukan figur manusia menjadi suatu sistem kerangka dengan menggunakan notasi Denavit dan Hartenberg
- Perhitungan posisi *end-effector* dengan menggunakan metoda Jacobian transpose.
- Penampilan kerangka manusia tersebut dengan menggunakan fungsi-fungsi OpenGL yang telah tersedia
- Pembuatan struktur dan rancangan antarmuka yang interaktif untuk menciptakan posisi-posisi figur manusia dalam frame.
- Proses penataan ruang editor dengan penyediaan empat ruang pandang yaitu tampak atas, tampak samping kanan, tampak depan dan tampak tiga dimensi dengan memakai fungsi-fungsi Library yang disediakan OpenGL
- Proses interpolasi dari frame-frame yang telah ada menjadi satu animasi

1.4. Pembatasan Masalah

Walaupun sebuah animasi seharusnya mencakup kerangka yang memiliki lapisan seperti daging, kulit bahkan baju, tetapi pembahasan masing-masing tersebut merupakan topik yang sangat luas sehingga pada tugas akhir ini tidak mencakup hal-hal diatas. Pembuatan perangkat lunak ini juga tidak menggunakan deformasi bentuk bebas (*free form deformation*) dan untuk figur yang dianimasikan

terbatas pada jenis figur manusia karena dianggap mampu memberikan gambaran yang lengkap untuk perkembangan figur-figur yang lain. Gerakan figur manusia ini dihitung terhadap satu titik pusat (*root*) yang dalam hal ini digunakan pelvis sebagai pusatnya. Semua masukan posisi *end-effector* akan dihitung terhadap pelvis, tidak memperhatikan unsur kecepatan maupun gravitasi dan rotasi joint dianggap terhadap satu sumbu saja yaitu sumbu x.

1.5. Sistematika Pembahasan

Pembahasan dalam Tugas Akhir ini dibagi dalam berbagai tahapan. Pembagian pokok bahasan disusun dengan metode induksi yaitu pembahasan mulai dari masalah-masalah yang umum, menuju ke masalah yang lebih spesifik, sesuai dengan tema bahasan.

Bab I berisi penjelasan mengenai latar belakang permasalahan, batasan masalah, tujuan dan sasaran pembahasan serta sistematika pembahasan keseluruhan Tugas Akhir ini.

Bab II berisi penjelasan mengenai teori-teori penunjang Tugas Akhir ini yaitu representasi figur menggunakan notasi Denavit Hartenberg (D-H notation), dasar dari kinematika, kinematika invers dengan menggunakan salah satu metode iteratif yaitu metode Jacobian Transpose , dan rumus interpolasi untuk menganimasikan frame-frame yang ada.

Bab III berisi penjelasan mengenai dasar-dasar dan konsep OpenGL. Dan penggunaannya dalam tugas akhir ini.

Bab IV diuraikan mengenai struktur data dan algoritma yang dipilih untuk mengimplementasikan konsep-konsep yang telah diuraikan dalam bab-bab terdahulu.

Bab V berisi penjelasan mengenai hasil uji kinerja perangkat lunak yang dibentuk berdasar Bab IV diatas.

Bab VI Berisi kesimpulan dari keseluruhan Tugas Akhir ini.

BAB II

KINEMATIKA INVERS

Dalam bab ini dibahas mengenai studi kepustakaan yang diperlukan untuk menyelesaikan masalah kinematika invers dan penggunaannya dalam animasi manusia. Terdapat dua metode utama yang sering digunakan dalam bidang robot untuk mengontrol struktur yang berartikulasi, yaitu kinematika dan dinamika. Kinematika adalah suatu ilmu tentang gerakan tanpa memperhatikan kekuatan yang menyebabkannya. Penekanannya hanya pada posisi, kecepatan dan percepatan relatif terhadap waktu. Kinematika dibedakan menjadi kinematika maju (*forward kinematics)* dan kinematika invers (*inverse kinematics*). Dalam kinematika maju diberikan data sudut untuk setiap joint lalu dilakukan perhitungan untuk menentukan posisi dan orientasi *end-effector*. Pada kinematika invers, diberikan posisi dan orientasi dari *end-effector* lalu dilakukan perhitungan untuk menentukan sudut dari setiap joint yang membentuk *end-effector* tersebut. Di lain pihak dinamika berhubungan dengan kekuatan, momen inersia dan massa dari setiap link.

2.1 Representasi Kinematika[1]

Sebuah figur yang berartikulasi adalah struktur yang terdiri dari sejumlah link yang terhubung oleh joint. Untuk menunjukkan hubungan antar joint ini terhadap joint yang lain diperlukan sebuah notasi kinematika. Salah satu jenis notasi yang

[1] Fu ,K.S. ,Gonzales, R.C. , Lee, C.S., "*Robotics*" , McGraw-Hill , New York, 1988

dapat digunakan adalah yang diperkenalkan oleh Denavit dan Hartenberg (D-H), yaitu dengan memberikan spesifikasi koordinat frame untuk setiap joint. Notasi D-H berisi sejumlah aturan untuk pemberian koordinat frame.

Untuk setiap link dapat ditetapkan sebuah sistem koordinat kartesian ortonormal ($\mathbf{x}_i$, $\mathbf{y}_i$, $\mathbf{z}_i$) pada sumbu jointnya, dimana i= 1,2,...,n (n=jumlah derajat kebebasan) ditambah dengan frame koordinat untuk *base*. Karena sebuah joint putar hanya memiliki satu derajat kebebasan, tiap frame koordinat lengan ($\mathbf{x}_i$, $\mathbf{y}_i$, $\mathbf{z}_i$) berkorespondensi dengan joint i+1 dan letak posisinya tetap pada link i. Ketika joint i aktif maka link i akan bergerak dengan acuan link i-1 . Karena sistem koordinat ke-i berada pada link i maka sistem koordinat tersebut akan ikut bergerak dengan link i. Sehingga untuk koordinat frame yang ke-n akan ikut bergerak dengan link n (*hand*). Koordinat *base* didefinisikan sebagai frame koordinat yang ke-0 (x_0, y_0, z_0). Masing-masing koordinat frame ditentukan dan ditetapkan berdasarkan tiga aturan, yaitu :

1. Sumbu z_{i-1} terletak sepanjang sumbu gerakan dari joint ke-*i*.
2. Sumbu x_i merupakan garis normal terhadap sumbu z_{i-1}.
3. Sedangkan sumbu y_i dapat ditentukan dari z_i dan x_i dengan aturan sistem koordinat tangan kanan.

Dengan aturan lokasi frame seperti diatas maka, koordinat ke-0 dapat ditentukan di sembarang tempat pada *supporting base*, asalkan sumbu z_0 terletak di sepanjang sumbu gerakan dari joint pertama. Frame koordinat terakhir (frame ke-n) dapat ditempatkan di sembarang tempat di tangan robot, selama sumbu x_n merupakan normal terhadap sumbu z_{n-1}.

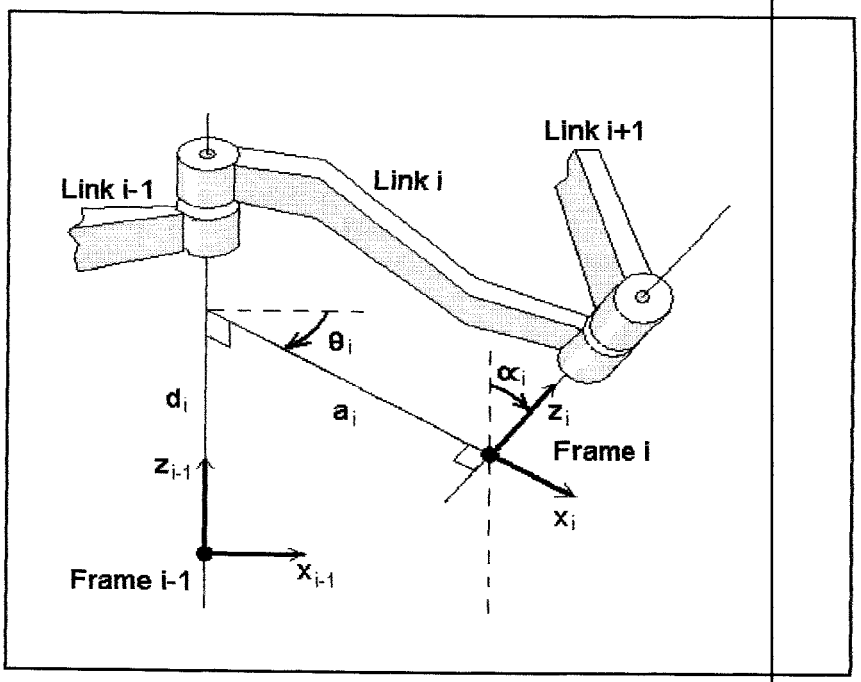
Representasi D-H dari sebuah link tergantung pada empat parameter geometris dari tiap link. Keempat parameter tersebut dapat didefinisikan sebagai berikut :

θ_i adalah sudut joint dari sumbu x_{i-1} ke sumbu x_i seputar sumbu z_{i-1} (menggunakan aturan tangan kanan)

d_i adalah jarak dari titik pusat frame koordinat ke-(i-1) ke perpotongan sumbu z_{i-1} dengan sumbu x_i sepanjang sumbu z_{i-1}

a_i adalah jarak offset dari perpotongan sumbu z_{i-1} dengan sumbu x_i ke titik pusat frame koordinat ke-i sepanjang sumbu x_i (atau jarak terpendek antara sumbu z_{i-1} dan z_i)

α_i adalah sudut offset dari sumbu z_{i-1} ke sumbu z_i seputar sumbu x_i (menggunakan aturan koordinat tangan kanan)



Gambar 2.1 Menentukan sistem koordinat dan parameternya

Untuk joint rotasi hanya d_i, a_i, dan α_i yang merupakan parameter joint yang berharga konstan, sementara θ_i merupakan variabel joint yang harganya berubah bila link i berputar pada joint *i*-1. Untuk joint prismatik θ_i a_i, dan α_i adalah parameter joint dan merupakan nilai konstan, sedangkan d_i merupakan variabel joint.

Dengan ketiga aturan untuk menetapkan sistem koordinat ortonormal di atas dapat dibuat prosedur untuk menetapkan sistem koordinat ortonormal untuk semua link dari sebuah robot melalui algoritma yang dijabarkan dalam beberapa langkah dibawah ini. Untuk robot dengan *n* derajat kebebasan :

1. Tetapkan sistem koordinat ortonormal tangan kanan (x_0,y_0,z_0) pada *supporting base* dengan sumbu z_0 terletak di sepanjang sumbu gerakan dari joint ke-1 dan menunjuk ke arah punggung lengan robot. Selanjutnya dapat ditetapkan sumbu x_0 dan y_0 yang merupakan normal terhadap sumbu z_0.
2. Untuk tiap *i*, *i*=1,2,...n-1 laksanakan langkah 3 sampai 6.
3. Tetapkan sumbu z_i pada sumbu putar joint *i*+1.
4. Letakkan titik pusat sistem koordinat joint ke-*i* pada titik potong antara sumbu z_i dan z_{i-1} atau titik potong antara sumbu z_i dengan normal antara z_i dan z_{i-1}.
5. Tetapkan $x_i = \pm(z_{i-1} \times z_i) / ||z_{i-1} \times z_i||$ atau sepanjang normal antara sumbu z_{i-1} dan z_i bila keduanya pararel.
6. Untuk melengkapi sistem koordinat tangan kanan di dapatkan $y_i = +(z_i \times x_i) / ||z_i \times x_i||$
7. Untuk menetapkan sistem koordinat tangan robot terlebih dahulu ditentukan sumbu x_n sepanjang dan searah dengan sumbu z_{n-1} dan arahnya menjauhi robot.

Tentukan sumbu x_n sedemikian rupa sehingga z_n merupakan normal dari sumbu z_{n-1} dan z_n. Tentukan y_n untuk melengkapi sistem koordinat tangan kanan.

8. Untuk tiap *i*, *i*=1,2,...n laksanakan langkah 9-12
9. Tentukan d_i yang merupakan jarak dari titik pusat sistem koordinat ke-(i-1) ke titik potong antara sumbu z_{i-1} dengan sumbu x_i sepanjang sumbu z_{i-1}
10. Tentukan a_i yang merupakan jarak dari titik potong antara sumbu z_{i-1} dengan sumbu x_i ke titik pusat sistem koordinat ke-i sepanjang sumbu x_i
11. Tentukan θ_i yang merupakan sudut rotasi dari sumbu x_{i-1} ke sumbu x_i seputar sumbu z_{i-1}. Merupakan variabel dari joint i.
12. Tentukan α_i yang merupakan sudut rotasi dari sumbu z_{i-1} ke z_i seputar sumbu x_i.

Setelah menentukan sistem koordinat untuk setiap link berdasarkan notasi D-H, maka dapat dibuat suatu matrik transformasi yang menghubungkan antara koordinat frame yang ke-i dengan koordinat frame yang ke i-1.

2.2 Persoalan Kinematika

Pada bahasan diatas telah diperlihatkan bahwa suatu struktur artikulasi/kerangka dapat dibuat dari kumpulan link yang dihubungkan dengan joint dan tersusun dalam suatu hierarki. Sebuah manipulator pada kerangka merupakan suatu rantai kinematika dari link dan joint. Dan dalam pembahasan ini akan diasumsikan bahwa joint adalah *rotary* yang berputar terhadap satu sumbu axis. Salah ujung dari sebuah manipulator, yaitu dasar (*base*) , adalah tetap dan tidak dapat bergerak. Sedangkan ujung yang lain bebas bergerak. Sebuah *end-effector*

dilekatkan pada koordinat frame dari joint yang terluar dari rantai. Posisi *end-effector* adalah titik di dalam frame ini.

Pada setiap joint dalam rantai didefinisikan sebuah transformasi M antara dua frame koordinat yang berada pada joint. Dari gambar 2.1 dapat diambil kesimpulan bahwa titik r_i yang berada pada koordinat frame ke-i dapat diekspresikan dalam koordinat frame yang ke i-1 sebagai r_{i-1} dengan melakukan transformasi-transformasi dibawah ini :

1. Rotasi sebesar θ_i terhadap sumbu z_{i-1} untuk menjajarkan sumbu x_{i-1} dengan sumbu x_i (sumbu x_{i-1} pararel terhadap x_i dan mempunyai arah yang sama)
2. Translasi sebesar d_i sepanjang sumbu z_{i-1} untuk membuat sumbu x_{i-1} dan x_i menjadi berhimpit.
3. Translasi sebesar a_i sepanjang sumbu x_i untuk membuat pusat sumbu juga berhimpit.
4. Rotasi sebesar α_i terhadap sumbu x_i untuk membuat kedua sistem koordinat menjadi berhimpit.

Setiap operasi diatas dapat diekspresikan dalam homogeneous matrik rotasi-translasi dan menghasilkan komposit matrik transformasi ${}^{i-1}A_i$ yang sering disebut matrik transformasi D-H untuk koordinat frame i dan i-1, yaitu :

${}^{i-1}A_i = T_{z,d}\, T_{z,\theta}\, T_{x,a}\, T_{x,\alpha} =$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a\cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{.....(2.1)}$$

Invers dari transfomasi ini adalah :

$$[{}^{i-1}A_i] = {}^{i}A_{i-1} = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a\cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{.....(2.2)}$$

Dengan menggunakan matrik ${}^{i-1}A_i$ suatu titik p_i yang terletak pada link i dan diekspresikan pada koordinat sistem i ke sistem koordinat i-1 dengan

$$p_{i-1} = {}^{i-1}A_i \, p_i \quad \text{.....(2.3)}$$

dimana

$p_{i-1} = (x_{i-1}, y_{i-1}, z_{i-1}, 1)^T$ dan $p_i = (x_i, y_i, z_i, 1)^T$

Matrik homogen ${}^{0}T_i$ yang menspesifikasikan lokasi koordinat frame i terhadap sistem koordinat *base* adalah hasil perkalian dari koordinat transformasi sebelumnya. Formulanya adalah :

$${}^{0}T_i = {}^{0}A_1 \, {}^{1}A_2 \, \ldots\ldots \, {}^{i-1}A_i = \prod_{j=1}^{i} {}^{j-1}A_j \qquad \text{untuk } i = 1,2,\ldots.n$$

$$= \begin{bmatrix} x_i & y_i & z_i & p_i \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^0R_i & {}^0p_i \\ 0 & 1 \end{bmatrix} \quad \text{.....(2.4)}$$

dimana

$[x_i \; y_i \; z_i]$ adalah matrik orientasi dari sistem koordinat yang ke-i yang terletak pada link i terhadap sistem koordinat frame *base*.

p_i adalah vektor posisi yang menunjuk dari pusat sistem koordinat *base* ke pusat sistem koordinat ke-i

2.3 Kinematika Maju dan Kinematika Invers

Apabila terdapat vektor **q** yang diketahui nilainya, maka problem dari Kinematika maju adalah menghitung vektor posisi dan orientasi **x** dari *end-effector* dengan persamaan

$$\mathbf{x} = f(\mathbf{q}) \quad \text{.....(2.5)}$$

Tetapi apabila tujuannya adalah meletakkan *end-effector* pada posisi dan orientasi **x**, maka untuk menentukan variabel joint vektor **q** memerlukan invers dari persamaan (2.5) yaitu :

$$\mathbf{q} = f^{-1}(\mathbf{x}) \quad \text{.....(2.6)}$$

Tidaklah mudah untuk memecahkan problem kinematika invers ini. Fungsi f adalah nonlinier, dan apabila pada persamaan (2.5) terdapat satu nilai **x** yang unik untuk **q** maka tidaklah sama dengan persamaan (2.6) dimana terdapat beberapa nilai **q** untuk sebuah nilai **x**.

Salah satu pendekatan masalah adalah dengan membuat problem menjadi linier terhadap konfigurasi manipulator. Sehingga dapat dikatakan hubungan antara joint *velocities* dan *velocity* dari *end-effector* adalah

$$\dot{x} = J(q)\dot{q} \qquad \text{.....(2.7)}$$

Dan hubungan liniernya diberikan dengan matrik *Jacobian*

$$J = \frac{\partial f}{\partial q} \qquad \text{.....(2.8)}$$

Yang menghubungkan perubahan variabel joint **q** pada perubahan posisi dan orientasi dari *end-effector* . **J** adalah matrik mxn , dimana n adalah jumlah variabel joint dan m adalah dimensi dari vektor **x** *end-effector* , yang biasanya bernilai 3 untuk penempatan posisi secara sederhana dan bernilai 6 untuk penempatan posisi serta orientasi yang lebih kompleks. Kolom **J** yang ke-*i* menunjukkan perubahan secara bertahap dalam posisi (dan orientasi) dari *end-effector* sebagai akibat dari perubahan terhadap variabel joint q_i. Dengan melakukan invers terhadap persamaan (2.7) akan menghasilkan

$$\dot{q} = J^{-1}(q)\dot{x} \qquad \text{.....(2.9)}$$

Setelah invers dari **J** diketahui maka dapat dilakukan perhitungan secara bertahap dari perubahan variabel joint terhadap perubahan dalam posisi dan orientasi dari *end-effector*[2]

[2] Watt, Allan "*Advanced Animation And Rendering Techniques*", Adison Wesley Publisher Ltd, New York, 1992

Metode iteratif untuk memecahkan kinematika invers didasari oleh persamaan (2.9) . Pada setiap iterasi $\dot{x}$ sebuah posisi $\dot{q}$ yang diinginkan dapat dihitung dari posisi *end-effector* yang sebelumnya. Perhitungan joint velocities dapat dilakukan menggunakan invers dari jacobian. Dan hasilnya dapat diintegrasikan untuk menentukan vector **q** yang baru. Prosedur ini dilakukan terus hingga posisi *end-effector* berada pada tempat yang diinginkan. Karena hubungan linier yang diwakili oleh **J** hanya valid untuk penyimpangan yang tidak terlalu banyak pada konfigurasi manipulator, maka perhitungan **J**(**q**) harus dilakukan pada setiap iterasi.

Pemecahan ini mempunyai kekurangan yaitu hanya dapat dilakukan apabila matrik Jacobian yang dihasilkan adalah matrik yang dapat diinvers yaitu *square* dan *non-singular.* Namun dalam kenyataannya hal ini tidaklah selalu terpenuhi. Untuk mengatasi ini digunakan rumus invers lain yaitu : *pseudoinverse*

Salah satu tujuan dari penelitian ini adalah untuk menyediakan manipulasi secara langsung pada figur yang berartikulasi. Dan untuk ini metode yang telah dibahas memiliki berbagai kekurangan. Perhitungan terhadap pseudoinverse memerlukan waktu komputasi yang berat. Untuk itu akan dibahas metode lain sebagai alternative yaitu : metode Jacobian Transpose.

2.4 Metode Jacobian Transpose

Diketahui gabungan tenaga tarikan **F** diberikan pada ujung dari sebuah manipulator, yang terdiri dari tarikan (*pull*) **f** pada suatu arah dan tenaga putaran (*torque*) **m** terhadap sumbu perputaran

$$\mathbf{F}=(f_x, f_y, f_z, m_x, m_y, m_z)^T \quad \text{.....(2.10)}$$

Apabila tenaga **F** ini dilaksanakan pada *end-effector* maka akan mengakibatkan terjadinya tenaga internal pada joint manipulator. Hubungan antara **F** dengan tenaga internal adalah :

$$\tau = \mathbf{J}^T\mathbf{F} \quad \text{.....(2.11)}$$

Hal ini menawarkan metode iteratif yang lebih sederhana untuk memaksa *end-effector* untuk berjalan melalui trayektori $X_d(t)$. Jika posisi *end-effector* pada saat ini diberikan dengan notasi $X_c(t)$, maka persamaan

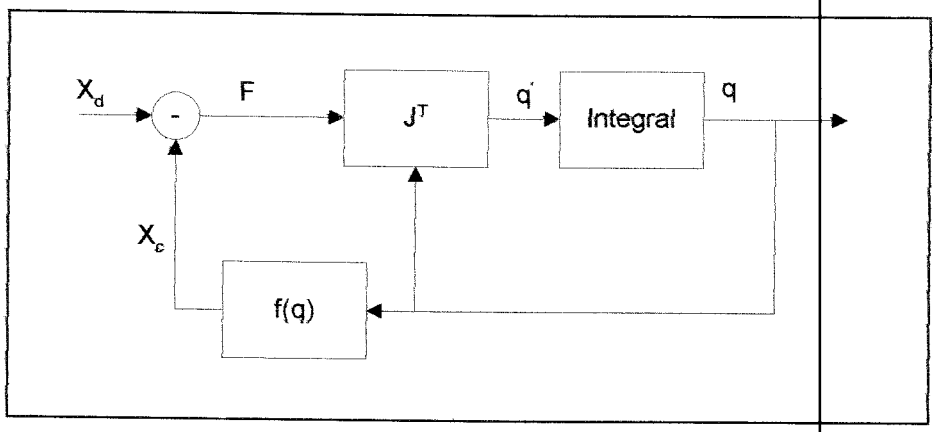
$$E(t) = X_d(t) - X_c(t) \quad \text{.....(2.12)}$$

dapat digunakan sebagai tenaga tarikan **f** yang menarik *end-effector* ke arah posisi trayektori $X_d(t)$. Persamaan (2.12) dapat digunakan untuk mentransformasi 'tenaga' external ini untuk menghasilkan tenaga pada setiap variabel joint. Di dalam tugas akhir ini ingin diketahui tanggapan dinamis dari sebuah manipulator terhadap tenaga yang diberikan maka τ dapat dianggap sebagai vektor dari variabel joint $\ddot{q}$. Dan karena tidaklah perlu untuk memasukkan unsur simulasi dinamik dari manipulator maka cukuplah menganggap τ sebagai vektor dari kecepatan penempatan joint. Sehingga persamaan (2.11) dapat ditulis sebagai berikut :

$$\dot{q} = J^T F \quad \text{.....(2.13)}$$

Setelah mendapatkan nilai $\dot{q}$ maka integrasi tunggal terhadap $\dot{q}$ akan menghasilkan vektor **q** yang baru, yang akan menggerakkan *end-effector* ke arah $X_d(t)$. Procedure ini berlanjut hingga *end-effector* mencapai posisi yang diinginkan[3].

Implementasi dari Metode Jacobian Transpose relatif lebih sederhana dibandingkan dengan metode sebelumnya. Pada setiap iterasi mencakup perhitungan terhadap tenaga tarikan yang diberikan kepada *end-effector*, perhitungan untuk matrik Jacobian untuk konfigurasi saat itu dan menghitung besarnya joint *velocities*. Hasilnya akan diintegrasi sebanyak satu kali untuk mendapatkan konfigurasi baru untuk manipulator. Gambar 2.2 di bawah ini menunjukkan looping dari proses ini.



Gambar 2.2 Model Looping untuk Metode *Jacobian Transpose*

Tenaga tarikan yang diberikan pada *end-effector* merupakan hasil interaksi dari pengguna.

---

[3] Welman, Chris , "*Inverse kinematics and Geometric Constraint For Articulated Figure Manipulation*", Simon Frasier University, 1993

2.4.1 Perhitungan Jacobian

Pada setiap iterasi , diperlukan perhitungan untuk mendapatkan matrik Jacobian **J** yang masing-masing kolomnya menunjukkan pergerakan frame dari *end-effector* dalam koordinat dunia sejalan dengan perubahan variabel joint. Apabila terdapat variabel vektor **q** dan posisi *end-effector* diwakili oleh posisi **P** dan orientasi **O** maka isi kolom jacobian untuk joint yang ke-i adalah :

$$J_i = \begin{bmatrix} \partial P_x \\ \partial P_y \\ \partial P_z \\ \partial O_x \\ \partial O_y \\ \partial O_z \end{bmatrix} \quad \text{.....(2.14)}$$

dimana operator derivasinya adalah terhadap q_i .

Pada setiap joint i dalam manipulator dapat melakukan translasi maupun rotasi terhadap salah satu axis $\mathbf{u}_i$ pada lokal koordinat framenya. Dan pada setiap joint $\mathbf{M}_i$ adalah matrik yang mentransformasikan lokal frame joint ke koordinat frame dunia. Apabila $\mathbf{axis}_i$ menunjukkan transformasi yang telah dinormalisasi dari axis lokal joint ke koordinat dunia, maka :

$$\mathbf{axis}_i = \mathbf{u}_i \, \mathbf{M}_i \quad \text{.....(2.15)}$$

dan untuk joint translasi entri dari Jacobian adalah :

$$J_i = \begin{bmatrix} [axis_i]^T \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{.....(2.16)}$$

sedangkan untuk joint rotasi entri dari Jacobian adalah :

$$J_i = \begin{bmatrix} [(p - j_i) \times axis_i]^T \\ [axis_i]^T \end{bmatrix} \quad \text{.....(2.17)}$$

Dimana p adalah posisi dari *end-effector* dan j_i adalah posisi joint I pada koordinat dunia.

2.4.2 Penskalaan

Dalam kenyataannya pada beberapa kasus model pada persamaan (2.13) tidak menghasilkan solusi yang memuaskan. Salah satu masalahnya adalah respon dari manipulator terhadap tenaga tarikan yang diberikan tergantung pada pemilihan skala pada joint variabel. Jika joint rotasi dalam degree akan menhasilkan respon nilai yang berbeda dibandingkan jika joint rotasi diukur dalam radian. Karena itulah digunakan skala untuk menyediakan tanggapan yang sama. Untuk menghasilkan skala yang stabil maka persamaan (2.13) dapat dimodifikasi menjadi :

$$\dot{q} = KJ^T F \quad \text{.....(2.18)}$$

dimana K adalah matrik skala konstan yang elemen diagonal ke – K_i bertindak sebagai faktor berat untuk joint ke-i. Nilai untuk K adalah :

$$K_i = \frac{1}{\omega_i \alpha_i} \quad \text{.....(2.19)}$$

dimana ω_i adalah panjang dari link i dan α_i adalah faktor berat untuk joint i.

2.4.3 Integrasi

Setelah nilai joint $\dot{q}$ diketahui, maka dengan menggunakan integrasi tingkat satu akan dihasilkan nilai **q** state yang baru untuk iterasi selanjutnya. Metode paling sederhana yang digunakan adalah Integrasi Euler tingkat satu yaitu :

$$q = q + h\dot{q} \quad \text{.....(2.20)}$$

Dengan menggunakan nilai pembeda integrasi h. Semakin kecil nilai h, maka akurasinya akan bertambah namun akan memperlambat proses yang dilakukan.

2.4.4 Batasan Joint

Karena constraint tidak diikutkan secara langsung pada persamman, maka dipergunakan limitasi pada variabel joint **q** . Yaitu dengan melakukan pemotongan nilai variabel ke batas atas dan batas bawah setelah proses integrasi.

2.5 Interpolasi

Pada bagian ini akan diterangkan metode untuk meng-interpolasi titik-titik join dari frame-frame yang ada, yaitu dengan menggunakan rumus interpolasi kurva NURBS.

Cara yang digunakan adalah menghasilkan kurva dengan menggunakan masukan dari titik data yaitu titik yang terletak pada kurva $\{\mathbf{Q}_i\}$, bukan titik kontrol kurva $\{\mathbf{P}_i\}$ seperti pada perhitungan kurva umumnya. Jika diinputkan sekumpulan titik $\{\mathbf{Q}_k\}$, $k = 0,\ldots,n$ dan ingin menginterpolasi titik tersebut dengan kurva dengan

pangkat p maka harus diterapkan nilai parameter $\{\bar{u}_k\}$ pada tiap $\mathbf{Q}_k$ dan memilih knot vektor yang tepat $U = \{u_0,\ldots,u_m\}$ sehingga dapat terbentuk sistem persamaan linear (n+1) x (n+1)

$$\mathbf{Q}_k = \mathbf{C}(\bar{u}_k) = \sum_{i=0}^{n} N_{i,p}(\bar{u})\mathbf{P}_i \qquad \text{.....(2.21)}$$

Titik kontrol Pi yang akan dicari sebanyak n+1. Masalahnya ada pada cara mencari $\bar{u}_k$ dan U, untuk $\bar{u}_k$ dipakai metode *chord length*

Jika d adalah total *chord length*

$$d = \sum_{k=1}^{n} |\mathbf{Q}_k - \mathbf{Q}_{k-1}| \qquad \text{.....(2.22)}$$

Maka

$$\left.\begin{aligned} \bar{u}_0 &= 0 \\ \bar{u}_n &= 1 \\ \bar{u}_k &= \bar{u}_{k-1} + \frac{|\mathbf{Q}_k - \mathbf{Q}_{k-1}|}{d} \qquad k = 1,\ldots,n-1 \end{aligned}\right\} \qquad \text{.....(2.23)}$$

Sedangkan cara mencari knot vektor U dengan menggunakan teknik *averaging* sebagai berikut :

$$\left.\begin{aligned} u_0 &= \ldots = u_p = 0 \\ u_{m-p} &= \ldots = u_m = 1 \\ u_{j+p} &= \frac{1}{p}\sum_{i=j}^{j+p-1} \bar{u}_i \qquad j = 1,\ldots,n-p \end{aligned}\right\} \qquad \text{.....(2.24)}$$

Dengan teknik *averaging* maka knot vektor *U* yang dihasilkan akan benar-benar mencerminkan distribusi dari $\bar{u}_k$. Rumus untuk mencari $\bar{u}_k$ akan selalu menghasilkan nilai yang positif sehingga untuk memecahkan persamaan linear diatas guna mencari P*i*

dapat dipecahkan dengan memakai metode *LU-Decompose* untuk memecah matriks Basis Function [N] menjadi dua buah matriks Lower[L] dan Upper[U] tanpa memakai *Pivoting*.

$$[N]\ [P] = [Q]$$

Matriks [N] dipecah dengan metode *LU-Decompose* menjadi matriks [L] (Lower) dan [U] (Upper)

$$[L]\ [U]\ [P] = [Q]$$

Selanjutnya digunakan *Forward Backward Substitution* untuk mendapatkan P sehingga

$$[L]\ (\ [U]\ [P]\ ) = [Q]$$

$$[L] \quad [Y] \quad = [Q]$$

$$[U]\ [P] = [Y]$$

2.6 Aplikasi Kinematika invers dalam Animasi Manusia

Pertama-tama yang harus dilakukan adalah menentukan benda apakah yang akan dianimasikan dan dalam tugas akhir ini yang dipilih adalah figur manusia. Walaupun secara ideal animasi terhadap manusia haruslah mengandung otot dan daging yang mengalami deformasi selama pergerakan, kulit dan baju yang mengkerut dan mengembang, rambut yang berkibar, expressi muka, serta

pewarnaan yang realistik. Tetapi semua atribut itu merupakan topik tersendiri yang mencakup bahasan yang luas. Dan penelitiannya masih dalam tahap eksperimen. Untuk saat ini pembahasan akan dibatasi pada tubuh manusia sederhana yang mendekati realita. Pendekatan sederhana ini dapat dianggap sebagai lapisan kerangka, yang dapat ditutup dengan otot, daging, dan kulit pada tahap berikutnya. Jadi permasalahannya dapat dibatasi pada control gerakan dari figur kerangka.

Sebuah kerangka dapat dibentuk dari kumpulan link yang dihubungkan bersama oleh joint. Jenis joint yang sering digunakan adalah joint rotasi atau translasi. Untuk merepresentasikan kerangka pada manusia yang digunakan adalah joint rotasi. Setiap joint rotasi dapat berputar dalam 1,2,3 arah ortogonal. Perputaran Ini menunjukkan derajat kebebasan (*DOF*) dari sebuah joint. Pendekatan secara detail dari kerangka manusia mampu mendekati hingga 200 *DOF* dan pemberian jarak jangkuan dapat dilakukan dengan pembatasan perputaran / rotasi dari joint-joint yang ada.

Kerangka yang lebih kompleks dapat dibentuk dengan mengatur link dalam suatu struktur tree. Transformasi yang berurutan dalam hierarki memastikan bahwa setiap node mewarisi rotasi yang dilakukan pada joint yang lebih tinggi posisinya dalam tree. Sebagai contoh rotasi yang dilakukan pada bahu akan mengakibatkan pergerakan seluruh lengan dan tidak hanya lengan atas. Salah satu joint perlu dipilih sebagai *root* dari tree. Dimana bila suatu transformasi dilakukan pada joint tersebut akan mengakibatkan pergerakan ke seluruh kerangka pada sistem koordinat.

2.7 Representasi Bentuk Badan Manusia [4]

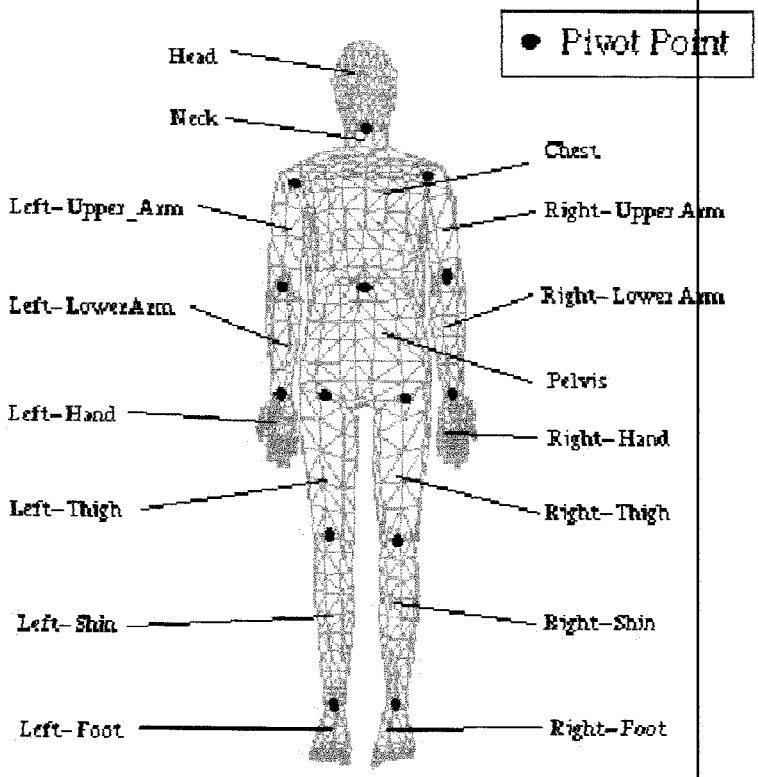
Pada bagian ini akan dijelaskan prosedur untuk merepresentasikan bentuk figur manusia. Termasuk didalamnya adalah pemberian frame koordinat dan melakukan sejumlah transformasi yang diperlukan untuk memanipulasi IK. Hal ini diperlukan untuk mengaplikasikan teori kinematika maju dan kinematika invers yang telah dibahas sebelumnya. Pemberian koordinat frame sangatlah penting, karena apabila terjadi kesalahan maka akan mengakibatkan error pada formulasi IK. Prosedur untuk pemberian lokal frame koordinat dimulai dengan penentuan titik pivot pada pusat dari frame koordinat untuk setiap joint yang ada (pelvis, bahu,siku dll) . Titik pivot ini dihitung relative terhadap koordinat dunia. Setelah itu barulah koordinat lokal dapat diberikan pada titik pivot tersebut dengan aturan notasi D-H. Pemberian titik pivot pada figur manusia dapat dilihat pada gambar 2.3

Perubahan terhadap suatu node akan menyebabkan node turunannya berubah juga. Kondisi ini terjadi ketika salah satu variabel joint dari suatu figur diubah melalui metode kinematika maju dan kinematika invers.

Untuk dapat membentuk figur manusia maka digunakan suatu keterangan yang menunjukkan posisi suatu joint pada figur tersebut. Keterangan ini memberikan informasi mengenai pergerakan joint relatif terhadap joint yang lain. Gambar 2.3 menunjukkan poligon mesh dari manusia dan nama dari setiap bagian dari bentuk badan manusia yang akan digunakan.

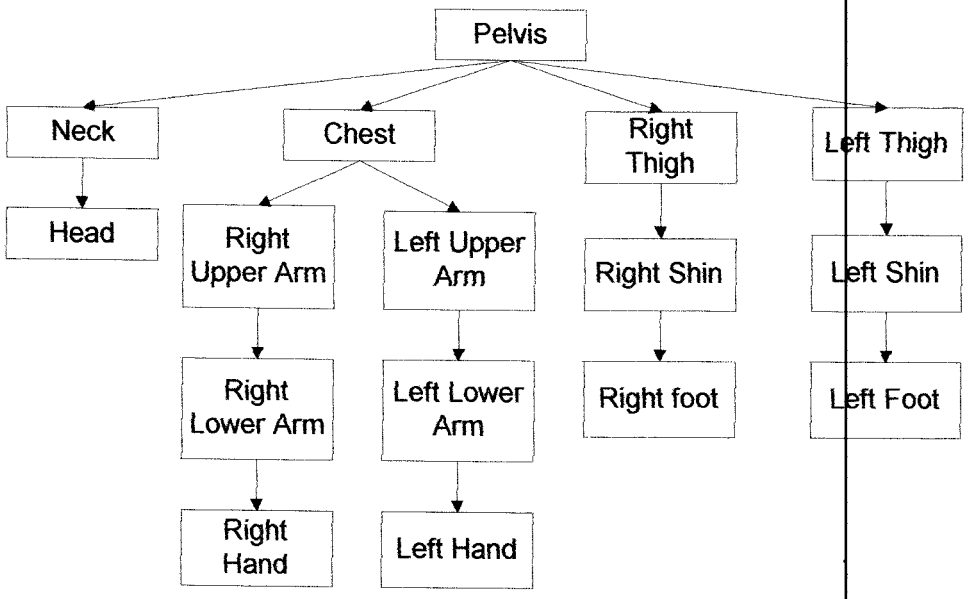
---

[4] Chin, Kwan. "*Closed Form And Generalized Inverse Kinematic Solutions for Animating Human Articulated Structure*", Curtin University Of Technology, 1996



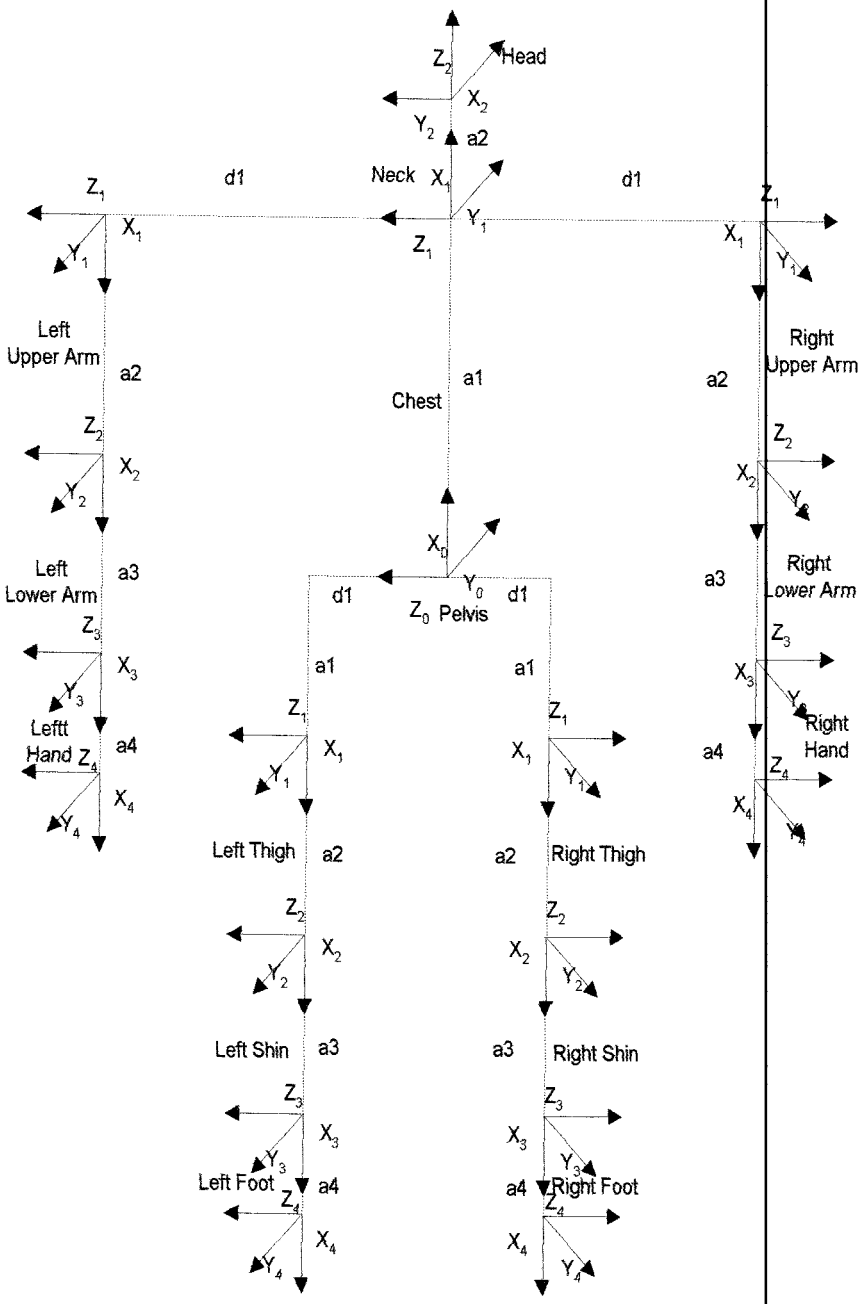
Gambar 2.3 Bagian figur manusia dan pemberian titik pivot

Dari bentuk manusia tersebut dapat dibuat suatu hirarki tree seperti pada gambar 2.4 yang berguna untuk menunjukkan urutan perubahan yang terjadi bila suatu joint mengalami perubahan. Sebagai contoh dengan menggunakan metode kinematika invers maka perubahan posisi tangan kanan akan mengakibatkan perubahan mulai dari pelvis, dada, tangan kanan atas, dan tangan kanan bawah. Sedangkan dengan metode kinematika maju maka perubahan sudut theta tangan kanan atas akan mengakibatkan perubahan posisi tangan kanan bawah dan tangan kanan.



Gambar 2.4 Hirarki Tree dari figur Manusia

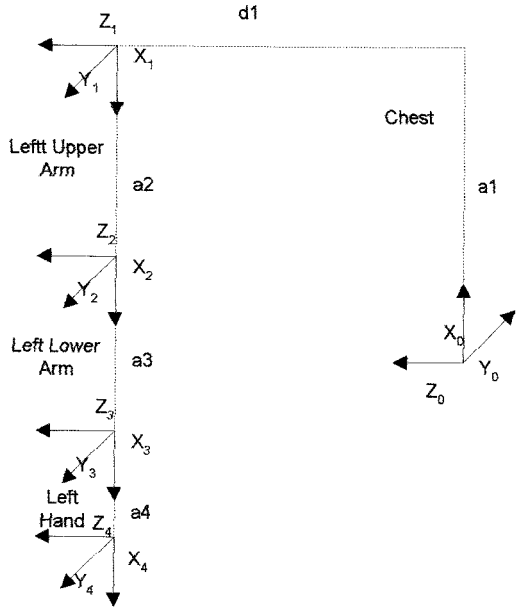
Berikut ini dijelaskan mengenai pemberian notasi D-H untuk manusia. Untuk tugas akhir ini digunakan notasi D-H untuk manusia seperti pada gambar 2.5 Seperti yang terlihat mada hirarki tree pada gambar 2.4 maka untuk figur manusia dibagi menjadi 5 bagian yaitu : bagian dari pelvis hingga tangan kiri (*Left hand*), bagian dari pelvis hingga tangan kanan(*Right hand*), bagian dari pelvis hingga kaki kiri (*Left Foot*), bagian dari pelvis hingga kaki kanan (*Right Foot*), dan pelvis hingga kepala (*Head*),.



Gambar 2.5 Notasi D-H untuk figur Manusia

Untuk mendapatkan nilai parameternya digunakan aturan D-H seperti yang telah dijelaskan pada bagian 2.1. Pada gambar 2.6 terlihat notasi D-H untuk bagian

tangan kiri . Parameter joint untuk Notasi D-H tangan kiri tersebut dapat dilihat pada tabel 2.1.



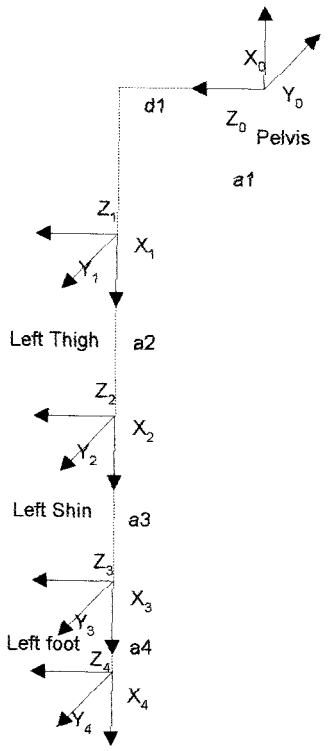
Gambar 2.6 Notasi D-H untuk tangan kiri

Tabel 2.1 Parameter D-H untuk tangan kiri

Joint	θ_i	d_i	α_i	a_i
1	180	3	0	$-a_1$
2	0	0	0	a_2
3	0	0	0	a_3
4	0	0	0	a_4

Parameter D-H untuk tangan kanan hampir sama dengan tangan kiri perbedaannya adalah pada sudut α_1 untuk tangan kanan bernilai 180.

Sedangkan pada gambar 2.7 adalah notasi D-H untuk bagian kaki kiri dan parameter joint untuk Notasi D-H kaki kiri tersebut dapat dilihat pada tabel 2.2



Gambar 2.7 Notasi D-H untuk kaki kiri

Tabel 2.2 Parameter D-H untuk kaki kiri

Joint	θ_i	d_i	α_i	a_i
1	180	d_1	0	a_1
2	0	0	0	a_2
3	0	0	0	a_3
4	0	0	0	a_4

Untuk menentukan semua posisi joint mulai dari *base* hingga *end-effector* digunakan metode kinematika maju. Penentuan posisi joint ini harus dilihat dari

frame koordinat acuan. Untuk posisi joint pada gambar 2.7 apabila diambil frame koordinat 0 sebagai *root*-nya maka posisi masing-masing joint dapat ditentukan sebagai berikut :

posisi joint pertama : $P_1 = {}^0A_1\, P_0$

posisi joint kedua : $P_2 = {}^0A_2\, P_0$

$= {}^0A_1\, {}^1A_2\, P_0$

Posisi joint ke-i : $P_i = {}^0A_i\, P_0$

$= {}^0A_1\, {}^1A_2 \ldots\ldots {}^{i-1}A_i\, P_0$

Tetapi apabila frame 4 sebagai *root* maka berarti semua posisi joint dihitung dari koordinat frame 4 sebagai frame base sehingga posisi masing-masing joint dapat ditentukan sebagai berikut :

posisi joint ketiga : $P_3 = {}^4A_3\, P_4$

posisi joint kedua : $P_2 = {}^4A_2\, P_4$

$= {}^4A_3\ {}^3A_2\, P_4$

Posisi joint ke-i : $P_i = {}^4A_i\, P_4$

$= {}^4A_3\, {}^3A_2 \ldots {}^1A_0\, P_4$ atau ${}^iA_{i-1} \ldots {}^1A_0\, P_i$

dimana P_i adalah posisi joint ke-i dan ${}^iA_{i-1} = ({}^{i-1}A_i)^{-1}$

apabila menggunakan koordinat 4 sebagai *root* maka koordinat yang dihasilkan harus dikembalikan menjadi frame koordinat 0. Ini dapat dilakukan dengan mengalikan semua koordinat hasilnya dengan matrik ${}^{i-1}A_i$.

Sebagai contoh adalah untuk bagian kaki kiri yang mempunyai notasi D-H seperti pada gambar 2.7 dan mempunyai parameter pada tabel 2.2. Maka posisi joint dengan pelvis sebagai frame koordinat 0 adalah sebagai berikut :

$$P_0 = [0\ 0\ 0\ 1]^T$$

$$P_1 = {}^0A_1\, P_0$$

$$= \begin{bmatrix} \cos 180 & -\cos 0 \sin 180 & \sin 0 \sin 180 & a_1 \cos 180 \\ \sin 180 & \cos 0 \cos 180 & -\sin 0 \cos 180 & a_1 \sin 180 \\ 0 & \sin 0 & \cos 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 0 & -a_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -a_1 \\ 0 \\ d_1 \\ 1 \end{bmatrix}$$

$$P_2 = {}^0A_2\, P_0$$

$$= {}^0A_1\, {}^1A_2\, P_0$$

$$= \begin{bmatrix} \cos 180 & -\cos 0 \sin 180 & \sin 0 \sin 180 & a_1 \cos 180 \\ \sin 180 & \cos 0 \cos 180 & -\sin 0 \cos 180 & a_1 \sin 180 \\ 0 & \sin 0 & \cos 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos 0 & -\cos 0 \sin 0 & \sin 0 \sin 0 & a_2 \cos 0 \\ \sin 0 & \cos 0 \cos 0 & -\sin 0 \cos 0 & a_2 \sin 0 \\ 0 & \sin 0 & \cos 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 0 & -a_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 0 & -a_2 - a_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & d_2 + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -a_2 - a_1 \\ 0 \\ d_2 + d_1 \\ 1 \end{bmatrix}$$

$$P_3 = {}^0A_3\, P_0$$

$= {}^{0}A_1\ {}^{1}A_2\ {}^{2}A_3\ P_0$

$= {}^{0}A_2\ {}^{2}A_3\ P_0$

$$= \begin{bmatrix} -1 & 0 & 0 & -a_2 - a_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & d_2+d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 0 & -\cos 0 \sin 0 & \sin 0 \sin 0 & a_3 \cos 0 \\ \sin 0 & \cos 0 \cos 0 & -\sin 0 \cos 0 & a_3 \sin 0 \\ 0 & \sin 0 & \cos 0 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 0 & -a_2 - a_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & d_2+d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 0 & -a_3 - a_2 - a_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & d_3+d_2+d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -a_3 - a_2 - a_1 \\ 0 \\ d_3+d_2+d_1 \\ 1 \end{bmatrix}$$

$P_4 = {}^{0}A_4\ P_0$

$= {}^{0}A_1\ {}^{1}A_2\ {}^{2}A_3\ {}^{3}A_4\ P_0$

$= {}^{0}A_3\ {}^{3}A_4\ P_0$

$$= \begin{bmatrix} -1 & 0 & 0 & -a_3 - a_2 - a_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & d_3+d_2+d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 0 & -\cos 0 \sin 0 & \sin 0 \sin 0 & a_4 \cos 0 \\ \sin 0 & \cos 0 \cos 0 & -\sin 0 \cos 0 & a_4 \sin 0 \\ 0 & \sin 0 & \cos 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 0 & -a_3 - a_2 - a_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & d_3+d_2+d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 0 & -a_4 - a_3 - a_2 - a_1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & d_4+d_3+d_2+d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -a_4 - a_3 - a_2 - a_1 \\ 0 \\ d_4+d_3+d_2+d_1 \\ 1 \end{bmatrix}$$

BAB III

KONSEP DASAR OPENGL

OpenGL adalah sebuah *library* untuk pemodelan dan grafik tiga dimensi yang mempunyai kelebihan dalam kecepatan dan dapat digunakan dimanapun. OpenGL bukanlah bahasa pemrograman tetapi merupakan API (*Application Programming Interface*). Apabila suatu program disebut *OpenGL-based* atau *OpenGL application* hal ini berarti bahwa program tersebut ditulis dalam suatu bahasa pemrograman (seperti C atau pascal) yang memanggil satu atau lebih fungsi dari *library* OpenGL.

Dalam bab ini dijelaskan mengenai konsep dasar dan fungsi-fungsi OpenGL diambil dari buku "*OpenGL Super Bible*" oleh Richard S. Wright Jr. , Michael Sweet dan buku "*OpenGL Programming Guide*" oleh Jackie Neider, Tom Davis.

Dengan menggunakan OpenGL, dapat dibuat suatu grafik 3D yang mendekati penggambaran dengan *ray tracing*. Walaupun demikian didalam OpenGL tidak terdapat perintah-perintah untuk menggambar benda tiga dimensi yang kompleks secara langsung. Untuk dapat membentuk benda-benda seperti mobil, bagian tubuh, maupun kapal selam dilakukan melalui sejumlah set perintah primitif seperti *points, lines* dan *polygons.*

3.1. Sintaks Perintah OpenGL

Semua perintah OpenGL mengikuti aturan penulisan yang memberi keterangan mengenai dari *library* mana fungsi tersebut berasal, berapa banyak

argumen yang diperlukan dan tipe datanya, serta keterangan mengenai apa yang dilakukannya. Format perintah OpenGL adalah :

<awalan library><perintah><optional jumlah argumen>< optional typeargumen>

Berikut adalah penjelasan mengenai perintah dasar OpenGL. Semua perintah dasar OpenGL menggunakan awalan gl diikuti dengan huruf kapital pada setiap kata membentuk nama perintah (sebagai contoh `glClearColor`). Untuk mendefinisikan konstanta diawali dengan GL_, menggunakan huruf kapital dan garis bawah untuk memisahkan kata (seperti `GL_COLOR_BUFFER_BIT`). Terkadang beberapa huruf dan angka ditambahkan pada akhir perintah (seperti *3f* pada `glColor3f`) .Dalam hal ini angka 3 menunjukkan berapa argument yang harus ada pada perintah tersebut dan akhiran huruf f menunjukkan jenis datanya yaitu floating.

Beberapa perintah OpenGL menerima sebanyak 8 macam tipe data yang berbeda, dengan membedakan akhiran pada perintahnya. Huruf yang digunakan sebagai akhiran untuk menspesifikasikan jenis datanya dapat dilihat pada tabel 3.1

Tabel 3.1 Jenis-jenis Tipe data dalam OpenGL

Akhiran	Tipe yang bersesuaian Dalam bahasa C	Definisi Tipe Dalam OpenGL
b 8-bit integer	signed char	GIbyte
s 16-bit integer	short	GIshort
i 32-bit integer	long	GLint, GIsize

f 32-bit Floating-point	float	GLfloat, GLclampf
d 64-bit Floating-point	double	GLdouble,GLclampd
ub 8-bit unsigned integer	unsigned char	GLubyte,GLboolean
us 16-bit unsigned integer	unsigned	GLushort
ui 32-bit unsigned integer	unsigned	GLuint,GLenum,GLbitfield

Sebagai contoh pada dua perintah berikut ini :

```
glVertex2i( 1, 3 ) ;
glVertex2f( 1.0, 3.0 ) ;
```

adalah sama yaitu meletakkan titik di layar pada koordinat x=1 dan y=2, perbedaannya perintah yang pertama menspesifikasikan titik dengan tipe data integer 32-bit dan yang kedua adalah dengan tipe data *single-precision floating point*.

Beberapa perintah OpenGL menembahkan huruf akhir *v* , yang menunjukkan bahwa perintah tersebut menggunakan argument pointer ke array / vektor . Di bawah ini contoh perbedaannya .

```
float color_array[]={1.0 , 0.0 , 0.0}
glColor3f(1.0 , 0.0 , 0.0) ;
glColor3fv(color_array);
```

3.2. Library yang Berhubungan dengan OpenGL

OpenGL menyediakan sejumlah set perintah untuk menggambar dan semua penggambaran yang lebih tinggi tingkatnya harus dilakukan dengan menggambil

fungsi dasar dari perintah ini. Maka dari itu dapat dibuat *library* sendiri diatas program OpenGL yang mempermudah pemrograman lebih lanjut. Fungsi asli dari OpenGL sendiri selalu dimulai dengan awalan *gl* terdapat pada *library* opengl32.dll dan file gl.h. Sedangkan beberapa *library* yang telah ditulis untuk menyediakan fungsi-fungsi tambahan pada OpenGL adalah :

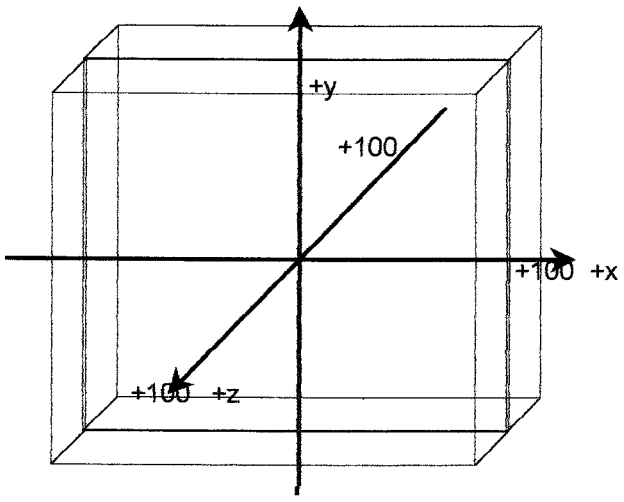
- *OpenGL Utility Library (GLU)* yang didalamnya terdapat sejumlah rutin yang menggunakan level bawah dari perintah OpenGL. Rutin-rutin ini mempunyai awalan *glu*. *Library* ini disediakan sebagai bagian dari implementasi OpenGL.
- *OpenGL Extension* untuk *X Window* yang menyediakan fungsi untuk menciptakan *OpenGL context* dan mengasosiasikannya dengan mesin yang menggunakan sistem *X Window*. Rutin-rutin ini mempunyai awalan *glx*.
- *Auxilary* atau *Aux Library*. Fungsi yang terdapat pada *library* ini sebenarnya bukanlah bagian dari OpenGL tetapi lebih merupakan sarana untuk menyediakan platform yang independen untuk pelaksanaan fungsi-fungsi OpenGL

3.3. Menggambar di Bidang Tiga Dimensi

Untuk mengambar jenis grafik apapun pada komputer biasanya dimulai dengan *pixel* . *Pixel* adalah elemen terkecil dari layar monitor yang mempunyai atribut warna dan posisi. Sedangkan untuk membentuk garis, poligon, lingkaran dll dapat dilakukan melalui urut-urutan pixel yang berbeda. Menggambar dengan menggunakan OpenGL mempunyai perbedaan dengan bahasa lain, yaitu : tidak perlu untuk memikirkan koordinat layar secara fisik tetapi hanya perlu untuk

menspesifikasikan posisi koordinat dalam volume penglihatan. OpenGl akan memikirkan sendiri cara bagaimana menggambar titik, garis dan lainnya yang berada dalam ruang 3D ke gambar 2D pada layar komputer.

Pada gambar 3.1 dibawah ini memperlihatkan suatu ruang pandang sederhana yang akan digunakan pada bagian ini. Area gambar yang dibatasi ini adalah ruang koordinat kartesian yang mempunyai *range* dari –100 hingga 100 untuk sumbu x,y, dan z. Secara sederhana ruang ini bisa dianggap sebagai bidang gambar untuk perintah-perintah OpenGL.



Gambar 3.1 Koordinat kartesian untuk volume pandang 100x100x100

Untuk menggambar titik digunakan suatu perintah OpenGL, yaitu : `glVertex`. Fungsi ini dapat mempunyai 2 sampai 4 parameter dari berbagai macam tipe data. Sebagai contoh perintah `glvertex` dibawah ini akan menspesifikasikan sebuah titik pada posisi 5 sumbu x, 5 sumbu y dan 0 untuk sumbu z.

```
glVertex3f (5.0f , 5.0f , 0.0f).
```

Sekarang setelah diketahui cara untuk menspesifikasikan sebuah titik di ruang pada OpenGL. Selanjutnya yang harus ditambahkan adalah informasi tambahan mengenai titik tersebut, apakah titik tersebut merupakan akhir dari sebuah garis, atau merupakan titik sudut dari sebuah poligon atau lainnya, karena definisi geometrik dari sebuah vertex sebenarnya bukanlah hanya sebuah titik pada layar tetapi lebih merupakan suatu titik dimana terjadi interseksi antara dua buah garis atau kurva.

Primitif adalah interprestasi sejumlah set atau deretan titik pada sebuah bentuk yang digambar pada layar. Pada OpenGL terdapat sepuluh macam primitif dari mulai menggambar sebuah titik hingga poligon. Untuk itu digunakan perintah glBegin sebagai cara memberi tahu OpenGL agar memulai menginterprestasikan sederetan titik sebagai salah satu bentuk primitif. Dan untuk mengakhiri deretan titik ini digunakan perintah glEnd. Sebagai contoh adalah dua perintah berikut ini :

perintah 1:

```
glBegin(GL_POINTS);                    //Menspesifikasikan titik sebagai primitif
      glVertex3f(0.0f,0.0f,0.0f);//Menspesifikasikan posisi suatu titik
      glVertex3f(5.0f,5.0f,5.0f);//Menspesifikasikan posisi suatu titik lain
glEnd();                    //Mengakhiri perintah menggambar titik
```

perintah 2:

```
glBegin(GL_LINES);                     //Menspesifikasikan garis sebagai primitif
      glVertex3f(0.0f,0.0f,0.0f );//Menspesifikasikan posisi titik awal garis
      glVertex3f(15.0f,15.0f,15.0f);//Menspesifikasikan posisi titik akhir
glEnd();                                    //Mengakhiri perintah menggambar titik
```

Pada perintah satu hasilnya berupa dua buah titik dilayar pada posisi (0.0f ,0.0f , 0.0f) dan (5.0f ,5.0f ,5.0f) sedangkan pada perintah kedua akan menghasilkan garis yang melalui titik (0.0f ,0.0f , 0.0f) , (5.0f ,5.0f ,5.0f) dan (15.0f ,15.0f , 15.0f) . Perlu diketahui bahwa jumlah perintah `glVertex` yang berada diantara `glBegin` dan `glEnd` tidak dibatasi. Tabel 3.2 dibawah ini adalah mengenai jenis primitif dan kegunaannya yang digunakan sebagai parameter dalam `glBegin`

Tabel 3.2 Primitif dari OpenGL yang digunakan sebagai argument dalam *glBegin*

Mode	Tipe Primitif
GL_POINTS	titik yang dispesifikasikan adalah titik-titik tunggal
GL_LINES	titik yang dispesifikasikan digunakan untuk menghasilkan garis. Setiap dua titik merupakan satu garis yang berdiri sendiri. Bila jumlah titiknya ganjil maka titik yang terakhir akan diabaikan
GL_LINE_STRIP	titik yang dispesifikasikan digunakan untuk menghasilkan garis bersambung. artinya setelah titik pertama maka titik selanjutnya merupakan tujuan perpanjangan garis. (Sama seperti perintah lineto pada bahasa C)
GL_LINE_LOOP	Sama seperti GL_LINE_STRIP,hanya terdapat tambahan garis yang menghubungkan antara titik pertama dan yang terakhir. Ini digunakan untuk menggambar daerah tertutup yang tidak mengikuti aturan GL_POLYGON
GL_TRIANGLES	titik yang dispesifikasikan digunakan untuk menghasilkan segitiga. Setiap tiga titik merupakan satu segitiga yang berdiri sendiri.
GL_TRIANGLES_STRIP	titik yang dispesifikasikan digunakan untuk sederetan

	segitiga. Setelah tiga titik pertama terbentuk setiap titik pada segitiga tersebut akan digunakan dengan dua titik berikutnya untuk membentuk segitiga yang lain.
GL_TRIANGLES_FAN	titik yang dispesifikasikan digunakan untuk menghasilkan segitiga yang berputar. Titik pertama berfungsi sebagai titik pusat dan setiap titik setelah titik yang ketiga akan digabung dengan berikutnya dan titik pusat.
GL_QUADS	Setiap empat titik digunakan untuk membentuk segiempat. Jika jumlah titiknya bukan kelipatan empat maka titik sisanya akan diabaikan
GL_QUAD_STRIP	titik yang dispesifikasikan digunakan untuk membentuk segiempat dengan aturan sebuah segiempat didefinisikan untuk setiap pasang titik setelah sepasang titik yang pertama.
GL_POLYGON	titik yang dispesifikasikan digunakan untuk membentuk suatu poligon. Sudut dari poligon tidak boleh berinterseksi. Titik yang terakhir secara otomatis akan dihubungkan dengan titik yang pertama.

Ketika menggambar titik tunggal, secara default size dari titik adalah satu. Untuk mengubah size ini digunakan dengan fungsi `glPointsize()` dengan parameter ukurannya. Selain mendefinisikan ukuran dari titik dapat juga dilakukan pengubahan lebar dari garis ketika melakukan penggambaran. Hal ini dapat dilakukan dengan perintah `glLineWidth()` yang mengambil parameter besarnya lebar garis.

3.4. Manipulasi dalam Ruang Tiga Dimensi

Dalam bagian ini akan dipelajari mengenai penentuan posisi pengamat dalam ruang, posisi benda dalam ruang, skala benda , penentukan transformasi perspektif dan melakukan transformasi matriks.

Proses transformasi untuk mendapatkan ruang yang diinginkan dapat dianalogikan dengan langkah-langkah di bawah ini :

1. Meletakkan tripod dan mengarahkan lensa kamera pada ruang (transformasi penglihatan)
2. Mengatur ruangan yang akan dipotret (transformasi pemodelan).
3. Memilih lensa kamera atau mengatur *zooming* (transformasi *proyeksi*)
4. Menentukan berapa besar hasil akhir dari foto , misalnya ukurannya dibuat lebih besar (transformasi *viewport*).

Untuk melakukan proyeksi dari koordinat 3D ke koordinat 2D di layar diperlukan transformasi. Dengan transformasi suatu benda dapat dirotasi, dipindahkan, maupun ditarik dan dikecilkan. Untuk memodifikasi benda yang dilakukan bukanlah mengubah benda secara langsung melainkan melakukan perubahan terhadap sistem koordinat. Apabila suatu koordinat telah dirotasi maka benda akan tampak terotasi ketika digambar. Di dalam OpenGL terdapat tiga macam transformasi yang perlu dilakukan sebelum melakukan penggambaran di layar yaitu penglihatan, pemodelan dan proyeksi. Penglihatan untuk menentukan posisi pengamat atau kamera, Pemodelan untuk memindahkan benda yang berada didalam ruang, dan proyeksi untuk meproyeksikan benda ke layar.

3.4.1. Transformasi Penglihatan dan Pemodelan

Di dalam OpenGL transformasi penglihatan dan pemodelan sangat berhubungan dan dikombinasikan menjadi satu matrik *modelview*.

3.4.1.1 Transformasi Penglihatan

Transformasi yang pertama kali dilakukan adalah penglihatan. Telah diketahui bahwa transformasi penglihatan dianalogikan dengan meletakkan dan mengarahkan kamera. Gunanya adalah untuk menentukan titik pandang yang menguntungkan. Secara default titik observasi berada pada posisi (0,0,0) dan melihat ke arah z negatif (ke arah dalam monitor). Titik observasi ini bergerak relatif terhadap sistem koordinat mata untuk menghasilkan titik pandang yang spesifik. Jika sebuah titik observasi berada pada origin maka sebua benda yang digambar dengan koordinat z positif akan berada di belakang pengamat.

Peletakkan titik observasi dapat dilakukan dimana saja dengan transformasi penglihatan. Transformasi Penglihatan harus dilakukan sebelum transformasi yang lainnya. Hal ini disebabkan transformasi penglihatan menggerakkan sistem koordinat yang sedang aktif terhadap koordinat mata. Sedangkan transformasi yang lainnya dapat dilakukan dengan didasarkan pada koordinat sistem yang baru.

Sebelum menspesifikasikan transformasi penglihatan maka matrik harus diinisialisai dengan menggunakan perintah `glLoadIdentity`. Hal ini perlu karena hampir semua perintah transformasi mengalikan matrik yang sedang aktif dengan matrik yang baru kemudian hasilnya akan menjadi matrik yang aktif. Jika tidak

menginisialisasi matrik yang sedang aktif maka itu berarti terus mengkombinasikan matrik transformasi yang sebelumnya dengan matrik transformasi yang baru.

Perintah yang digunakan untuk mendefinisikan transformasi penglihatan adalah :

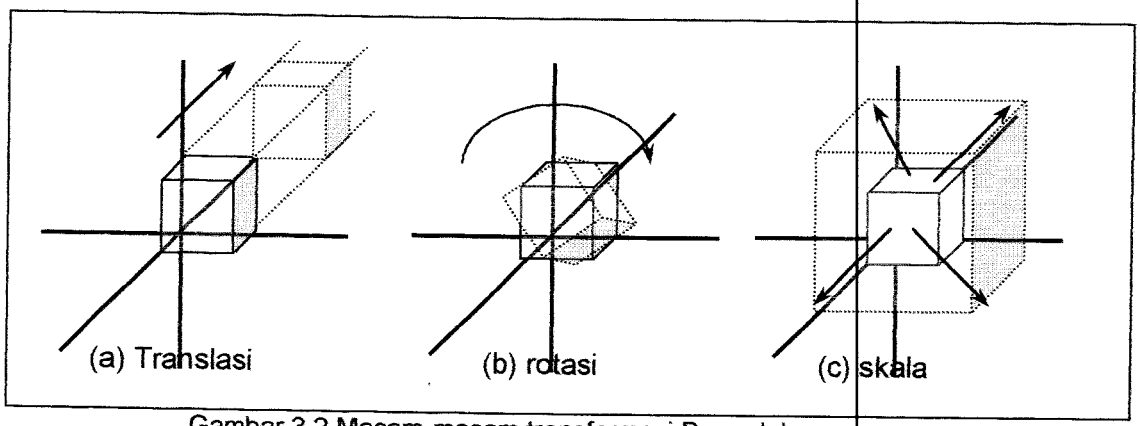
```
gluLookAt( glDouble eyeX, glDouble eyeY, glDouble eyeZ,
           glDouble centerX,glDouble centerY,glDouble centerZ,
           glDouble upX, glDouble upY, glDouble upZ)
```

dimana :

eyeX, eyeY, eyeZ	: adalah posisi dari pengamat
centerX, centerY, centerZ	: koordinat pusat ruang yang akan diamati
upX, upY, upZ	: untuk menspesifikasi *up vector.*

3.4.1.2 Transformasi Modeling

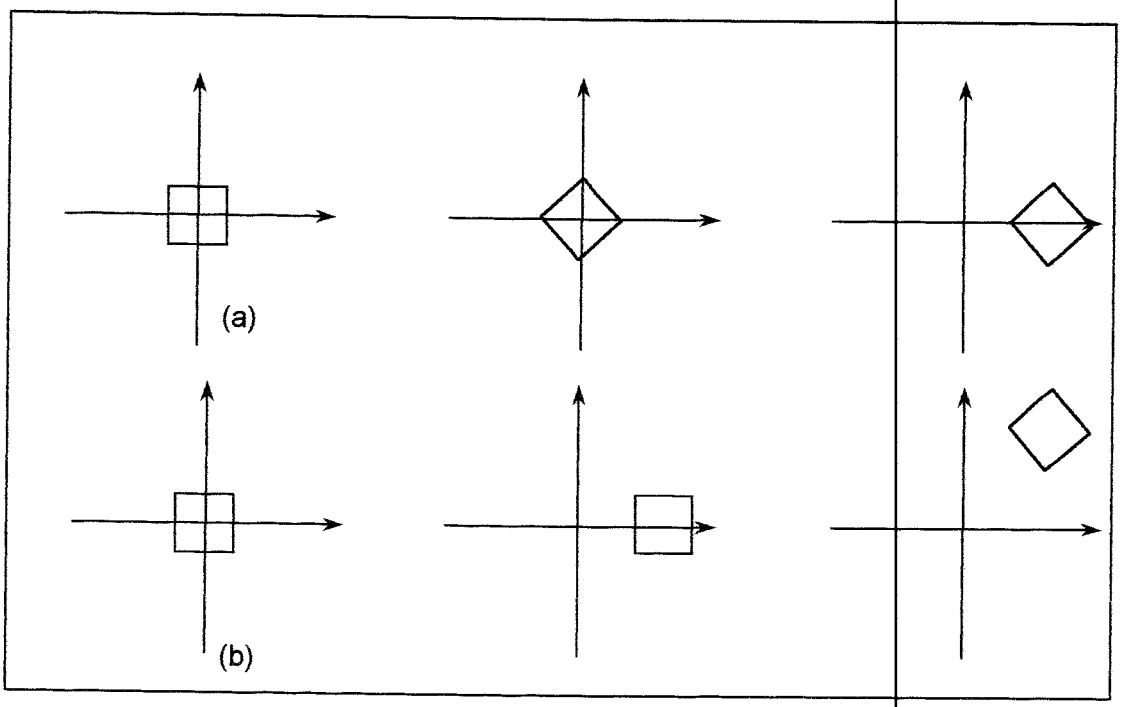
Transformasi pemodelan digunakan untuk memanipulasi benda. transformasi ini menggerakkan benda ke suatu tempat, memutar dan membuatnya sesuai dengan skala. Gambar 3.2 memperlihatkan tiga macam transformasi pemodelan yang dapat dilaksanakan pada benda.



Gambar 3.2 Macam-macam transformasi Pemodelan

Pada gambar 3.2a Memperlihatkan translasi, dimana benda dipindahkan sepanjang sumbu yang ditentukan. Gambar 3.2b memperlihatkan benda dirotasi terhadap salah satu sumbu. Dan pada gambar 3.2c terlihat bahwa dimensi benda ditambah dengan suatu angka. Skala dapat digunakan untuk *nonuniform* dimensi, dimana suatu benda dapat nampak ditarik maupun dikecilkan.

Hasil akhir yang tampak pada layar atau benda sangatlah tergantung pada urutan transformasi pemodelan yang dilakukan. Hal ini dapat diilustrasikan dalam gambar 3.3.



Gambar 3.3 Perbedaan hasil transformasi Pemodelan

Pada gambar 3.3(a) memperlihatkan tahapan proses rotasi terlebih dahulu terhadap suatu segiempat pada sumbu z yang dilanjutkan dengan translasi ke arah sumbu x yang sudah dirotasi. Sedangkan pada gambar 3.3(b) segiempat yang sama

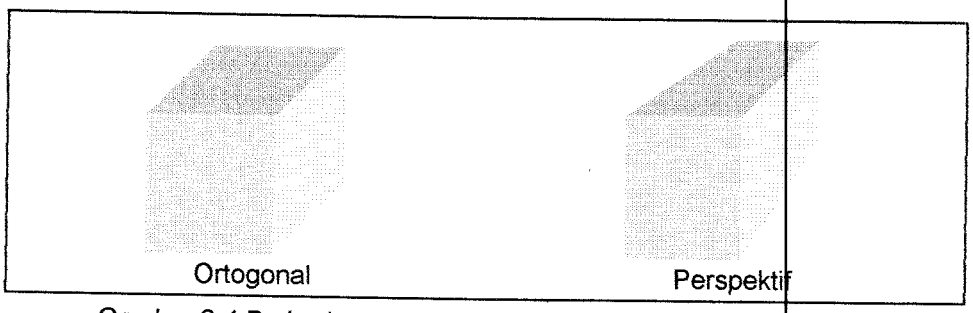
ditranslasi terlebih dahulu ke arah sumbu x positif baru kemudian dirotasi terhadap sumbu z.

3.4.2. Transformasi Proyeksi

Transformasi Proyeksi dilakukan pada hasil dari transformasi pemodelan. Proyeksi ini mendefinisikan volume pandangan dan *clipping plane*. Secara spesifik transformasi *proyeksi* menyatakan bagaimana ruang yang telah selesai (setelah semua pemodelan selesai) diterjemahkan ke gambar final pada layar. Terdapat dua macam proyeksi yaitu : ortogonal dan perspektif.

Di dalam proyeksi ortogonal semua poligon digambar pada layar dengan tepat tanpa mengubah ukuran dari benda yang sesungguhnya. Dan proyeksi inilah yang sering dipakai dalam desain arsitek dan CAD.

Proyeksi Perspektif adalah proyeksi yang meperlihatkan benda seperti pada kenyataannya yang terlihat pada kehidupan nyata. Yang menjadi ciri dari proyeksi perspektif adalah *foreshortening* yang membuat benda yang terletak lebih jauh menjadi lebih kecil daripada benda yang dekat. Sehingga bagian benda yang letaknya jauh dari pengamat akan terlihat lebih kecil. Dan garis pararel tidak selalu digambar dengan pararel pula. Untuk gambar kubus misalnya , sisi kubus adalah pararel tetapi dalam proyeksi perspektif sisi tersebut tampak mengecil ke suatu titik. Titik ini disebut titik hilang (*vanishing point*). Pada gambar 3.4 ini bawah ini terlihat perbedaan antara proyeksi ortogonal dan perspektif.

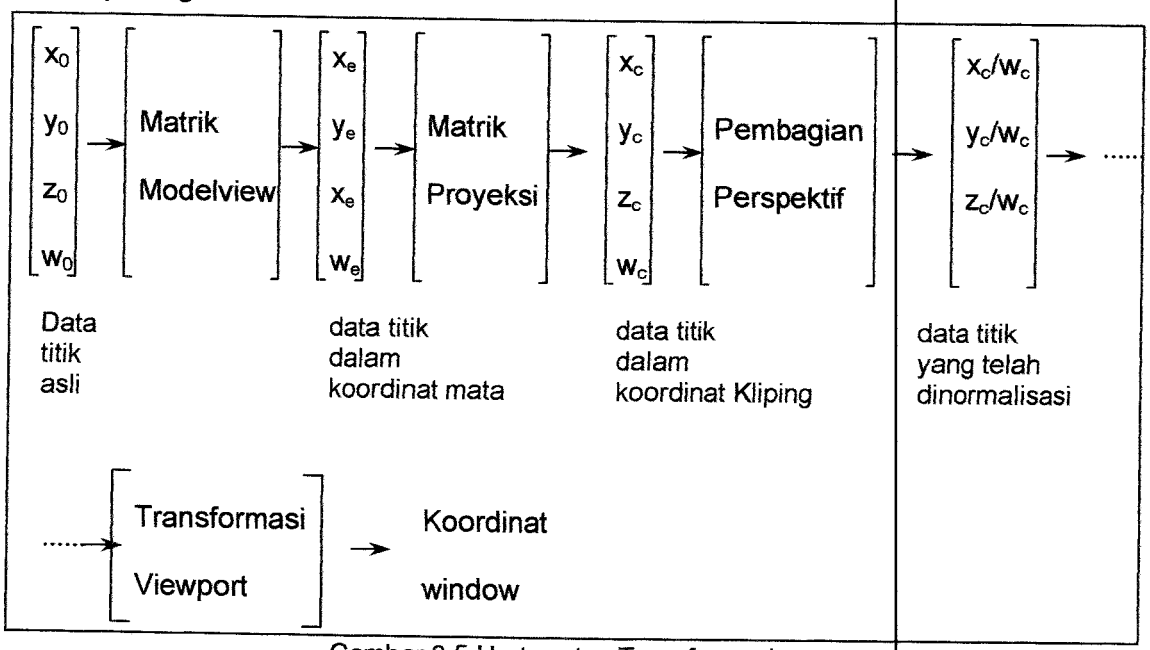


Gambar 3.4 Perbedaan antara proyeksi ortogonal dan perspektif

3.4.3. Matrix Dalam OpenGL

Setelah perkenalan terhadap dasar dan definisi dari transformasi maka pada bagian ini akan dibahas elemen matematika dibalik transformasi ini. Setiap transformasi yang telah dibahas dapat diperoleh dengan mengalikan matris yang berisi titik dengan matriks yang berisi transformasi. Semua transformasi yang ada di dalam OpenGL dapat dijelaskan sebagai perkalian dua atau lebih matrik.

Urut-urutan transformasi yang dilakukan untuk dapat menggambar di layar terlihat pada gambar 3.5.



Gambar 3.5 Urut-urutan Transformasi

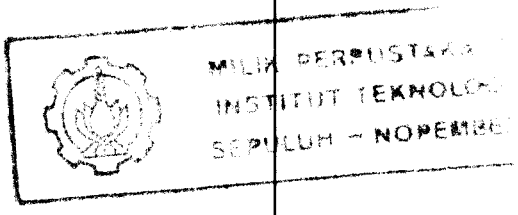
Pertama kali data titik pada ruang 3D dikonversikan menjadi matrik 1x4 dimana tiga elemen pertama adalah koordinat x,y dan z. Elemen yang keempat adalah faktor skala dengan 1 sebagai nilai default. Titik ini kemudian dikalikan dengan matrik *modelview* yang menghasilkan koordinat mata yang telah ditransformasi. Setelah itu koordinat mata akan dikalikan dengan matrik proyeksi untuk menghasilkan koordinat klipping. Nilai w dapat diubah oleh matrik proyeksi atau matrik *modelview*. Dan tahap terakhir adalah melakukan mapping dari koordinat 3D ke koordinat 2D dengan transformasi *viewport*. Transformasi ini juga direpresentasikan dengan matrik tetapi isi matrik tersebut tidak diisi secara langsung oleh pengguna. OpenGL akan menspesifikasikannya tergantung pada parameter yang dimasukkan pada perintah `glViewport`. Untuk menspesifikasikan jenis matrik yang sedang diproses digunakan perintah `glMatrixMode(Glenum mode);` mode yang digunakan dapat `GL_MODELVIEW` maupun `GL_PROJECTION`

3.4.4. Matrik *Modelview*

Matrik *modelview* adalah matrik berukuran 4x4 yang merepresentasikan sistem koordinat yang telah ditransformasi. Sama seperti pada perintah glViewport isi dari matrik modelview ini tidak disi secara langsung karena OpenGL merepresentasikannya langsung tergantung pada translasi,rotasi, dan skala yang diberikan.

Translasi

Perintah yang digunakan adalah : `glTranslatef( Glfloat x, Glfloat y, Glfloat z);`



Fungsi ini mengambil tiga parameter yaitu banyaknya translasi kearah sumbu x, sumbu y dan sumbu z.

Rotasi

Perintah yang digunakan adalah : `glRotatef(Glfloat angle, Glfloat x, Glfloat y, Glfloat z);`

Disini akan dilakukan perputaran terhadap vektor yang dispesifikasikan oleh nilai argumen x, y, dan z dengan angle sebagai nilai derajat besarnya perputaran. Arah perputaran adalah *counterclockwise*

Skala

Perintah yang digunakan adalah : `glScalef(Glfloat x,Glfloat y, Glfloat z);`

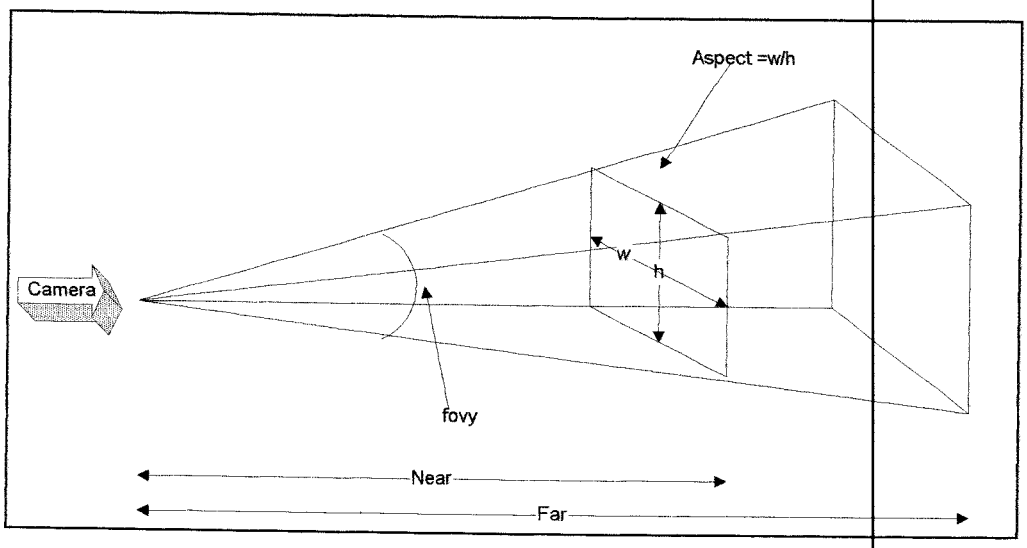
Dengan skala akan memperbesar/memperkecil benda sepanjang sumbu yang dispesifikasikan dengan cara mengalikan nilai x,y dan z benda dengan masing-masing argumen. Skala tidak selalu harus uniform, dengan perintah ini dapat dilakukan *stretching* maupun *squeezing* pada benda .

3.4.5. Matrik Proyeksi

Setting terhadap matrik proyeksi dapat dilakukan dengan dua cara tergantung pada jenis proyeksi yang digunakan.

3.4.5.1 Perspektif

Penjelasan mengenai proyeksi perspektif telah dijelaskan pada bagian 3.4.42 Ilustrasi mengenai proyeksi ini dalam OpenGL dapat dilihat pada gambar 3.6 berikut ini.



Gambar 3.6 Ilustasi Proyeksi perspektif dalam OpenGL

Perintahnya adalah

```
glMatrixMode(GL_PROJECTION)
glLoadIdentity;
gluperspective(Gldouble fovy, Gldouble aspect, Gldouble znear,
Gldouble zfar)
```

dimana :

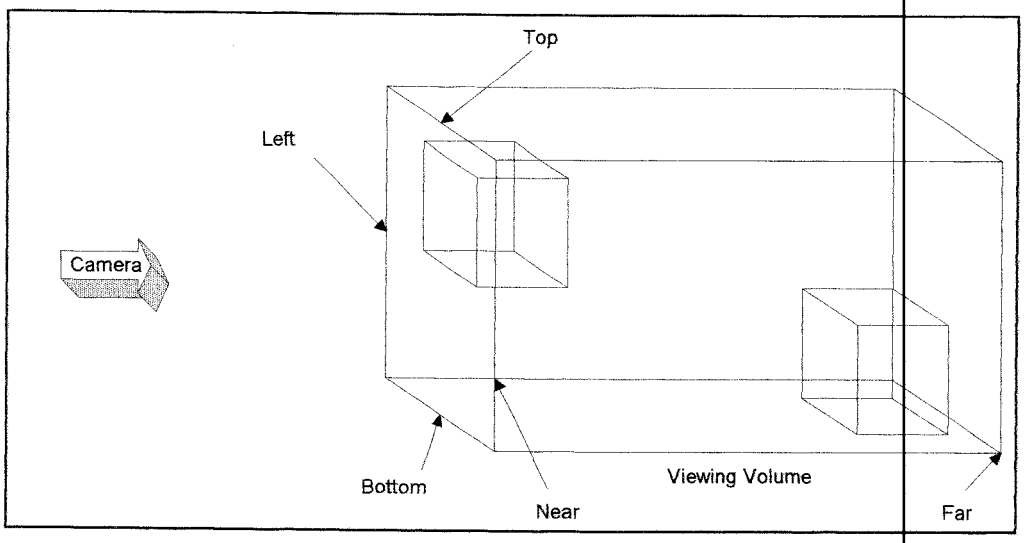
fovy = sudut pandangan pada bidang x-z. Nilainya harus antara 0°-180°

aspect ratio = width dibagi height

znear dan *zfar* = jarak antara titik pengamat dengan clipping plane sepanjang sumbu z dan nilainya selalu positif

3.4.5.2 Ortogonal

Penjelasan mengenai proyeksi Ortogonal juga telah dijelaskan pada bagian 3.4.42 Ilustrasi mengenai proyeksi ini dalam OpenGL dapat dilihat pada gambar dibawah ini.



Gambar 3.7 Ilustasi Proyeksi Ortogonal dalam OpenGL

Perintahnya adalah

```
glMatrixMode(GL_PROJECTION)
glLoadIdentity;
 GlOrtho   (Gldouble left, Gldouble right , Gldouble bottom,
            Gldouble top, Gldouble near , Gldouble far )
```

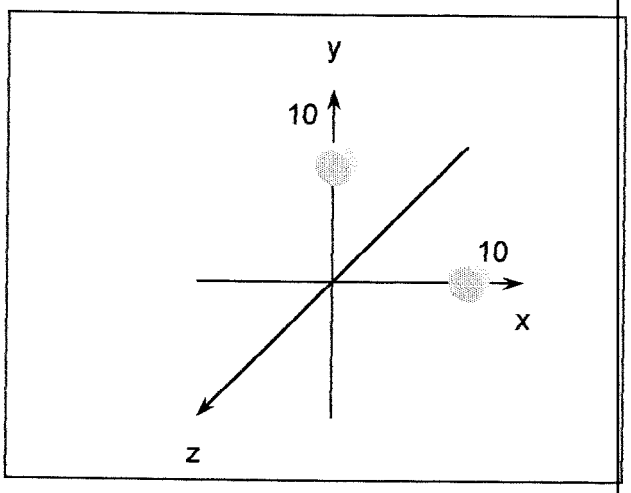
3.4.6. Matrik Indentitas

Untuk mereset isi matrik *modelview* ke keadaan awal, yaitu sebelum dirotasi, translasi maupun skala, dapat digunakan perintah `glLoadIdentity();`

Berikut adalah contoh kode program yang mengilustrasikan penggunaan dan *setting* terhadap matrik *modelview*

```
//setting matrik menjadi modelview dan mereset isinya
glMatrixMode(GL_MODELVIEW)
glLoadIdentity();

//Pindah 10 unit ke arah sumbu y positif dan gambar bola pertama
glTranslatef(0.0f, 10.0f, 0.0f);
auxSolidSphere(1.0f);

// reset isi matrik modelview
glLoadIdentity();
//Pindah 10 unit ke arah sumbu x positif dan gambar bola kedua
glTranslatef(10.0f, 0.0f, 0.0f);
auxSolidSphere(1.0f);
```

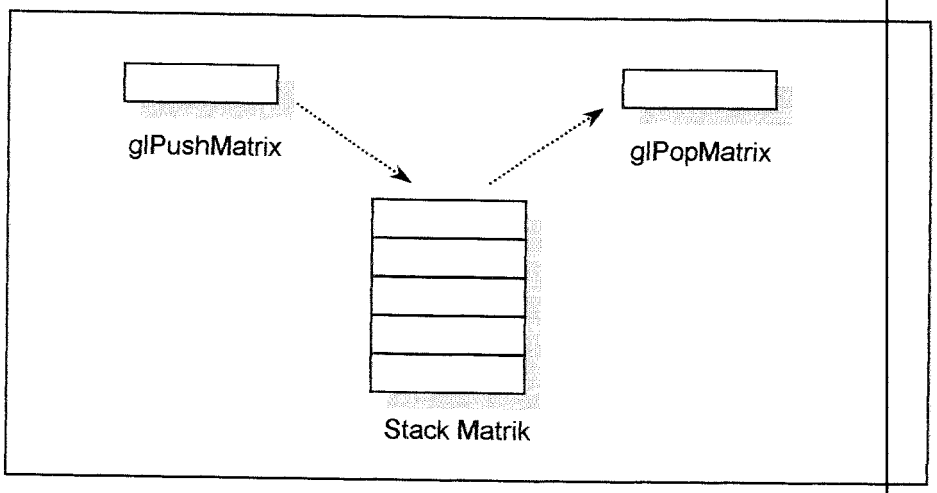
Hasil dari kode di atas adalah pada gambar 3.8 berikut ini



Gambar 3.8 Hasil proses translasi

3.4.7. Matrik Stack

Dalam penggambaran terkadang diinginkan untuk dapat menyimpan status transformasi sekarang dan mereload kembali setelah menggambar suatu benda, daripada mereset matrik lalu melakukan *setting* ulang lagi. Untuk merealisasikan hal ini OpenGL menyediakan stack matrik untuk matrik *modelview* dan proyeksi. Sebuah matrik stack bekerja seperti pada program stack biasa, yaitu dapat dilakukan perintah *push* untuk menyimpan matrik yang sedang aktif ke dalam stack untuk menyimpannya dan perintah *pop* untuk mengembalikan nilai. Kedua perintah ini dalam OpenGL adalah : `glPushMatrix()` dan `glPopMatrik();`



Gambar 3.9 Ilustrasi Kerja Matrik Stack

nilai maksimum yang dicapai untuk kedalaman stack didapat dari perintah

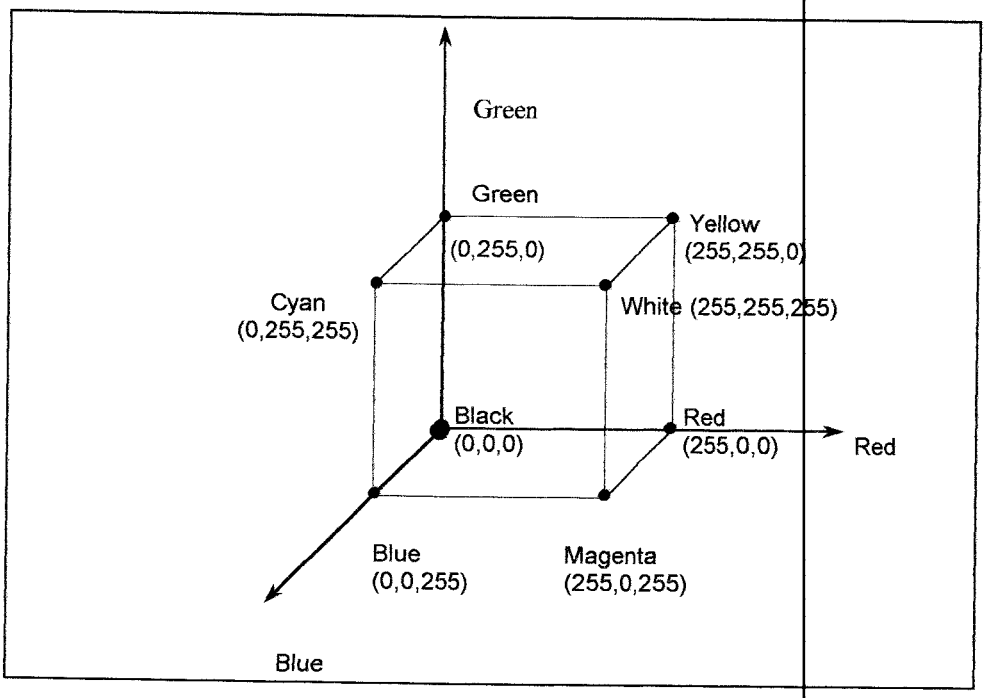
`glGet(GL_MAX_MODELVIEW_STACK_DEPTH);` atau

`glGet(GL_MAX_PROJECTION _STACK_DEPTH);`

Pesan error yang muncul apabila ingin memasukkan matrik ke dalam stack yang sudah penuh adalah `GL_STACK_OVERFLOW` dan apabila melakukan perintah pop pada stack yang kosong maka pesan errornya adalah `GL_STACK_UNDERFLOW`.

3.5. Pewarnaan

OpenGl menspesifikasikan warna sebagai gabungan intensitas komponen merah, hijau dan biru. Berdasarkan ini maka dapat dibentuk suatu ruang warna RGB yang menunjukkan kombinasi warna yang dapat digunakan. Gambar 3.7 dibawah ini menunjukkan warna sebagai ruang dengan merah, hijau dan biru sebagai sumbunya.



Gambar 3.10 Ruang warna RGB

Untuk mengeset warna dapat dilakukan dengan perintah :

```
glColor<x><t>(red, green, blue, alpha)
```

dimana

<x> menunjukkan jumlah argumen, dapat bernilai 3 untuk tiga argumen red, green, blue atau 4 argumen dengan tambahan komponen alpha

<t> menunjukkan tipe data argumen dapat berupa b , d , f , I , s , ub , ui , us untuk *byte, double, float, integer, short, unsigned byte, unsigned integer,* dan *unsigned short.*

3.6. Pencahayaan

Dalam bagian ini akan dibahas mengenai bagaimana cara mengatur model kamera, parameter, dan mengatur pantulan material.

3.6.1. Cahaya

Di dalam OpenGL terdapat tiga macam cahaya yaitu : *ambient*, *diffuse*, dan *specular.*

Cahaya *Ambient* adalah cahaya yang arah datangnya tidak tentu. Walaupun mempunyai sumber tetapi sinarnya telah dipantulkan oleh ruangan (dinding, bedda-benda lainnya) sehingga arah datangnya tidak bisa ditentukan lagi.

Cahaya *Diffuse* datang dari arah tertentu tetapi dipantulkan sama rata oleh permukaan benda. Walaupun cahaya dipantulkan sama rata tetapi permukaan benda akan tampak lebih terang apabila mendapatkan cahaya searah garis lurus daripada cahaya yang datangnya dari suatu sudut.

Cahaya *Specular* seperti pada cahaya *diffuse*, cahaya specular mempunyai arah tetapi pemantulannya lebih tajam dan arahnya tertentu.

Tidak ada jenis sumber cahaya yang seleuruhnya terbuat dari satu elemen diatas, melainkan merupakan variasi dari ketiganya.

3.6.2. Material

Selain mendapat cahaya dari luar , sebuah benda mempunyai warnanya sendiri-sendiri. Pada bagian yang sebelumnya sebuah warna benda adalah berdasarkan cahaya yang dipantulkan oleh benda tersebut. Bola berwarna biru sebenarnya merupakan bola yang disinari oleh sinar putih dan bola tersebut memantulkan warna biru dan menyerap warna yang lain. Dengan cahaya putih maka semua benda akan tampak dengan warna naturalnya masing-masing. Ketika menggunakan pencahayaan. suatu benda dideskripsikan sebagi material yang mempunyai properti pemantulan tersendiri. Sebagai contoh : Suatu benda berwarna merah dinyatakan terbuat dari material yang memantulkan sebagian besar cahaya merah. Tetapi harus juga diingat bahwa seperti pada cahaya yang datang maka cahaya yang dipantulkan juga dapat memiliki nilai *ambient, diffuse* dan *specular*. Sebuah material dapat memiliki nilai kilau dan memantulkan cahaya specular dan menyerap cahaya ambient dan diffuse.

3.6.3. Pengaturan Cahaya dalam Ruang

Untuk memerintahkan OpenGL agar melakukan perhitungan cahaya dapat digunakan fungsi `glEnable` dan parameternya adalah `GL_LIGHTING` seperti berikut :

```
glEnable(GL_LIGHTING)
```

Fungsi ini memberitahu OpenGL agar menggunakan properti material dan cahaya untuk menentukan warna dari setiap titik pada layar.

Setelah mengenable perhitungan untuk cahaya maka berikutnya adalah menentukan model cahaya. Tiga parameter yang mempengaruhi model cahaya akan diseting dengan fungsi *glLightModel()*.

Parameter pertama yang digunakan untuk contoh dibawah ini adalah GL_LIGHT_MODEL_AMBIENT.

```
// Cahaya putih
glFloat ambientLight[]={1.0f, 1.0f, 1.0f, 1.0f);
//enable cahaya
glEnable(GL_LIGHTING);
//Seting model cahaya untuk menggunakan cahaya ambient yang dispesifikasi oleh
//ambientLight[]
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientLight);
```

Salah satu variasi dari fungsi glLightModel adalah *glLightModelfv* yang mengambil parameter pertama sebagai model cahaya yang akan diset. Dibawah ini adalah variasi dari fungsi glLightModel.

```
void glLightModelf(Glenum pname, Glfloat param);
void glLightModeli(Glenum pname, Glint param);
void glLightModelfv(Glenum pname, const Glfloat *params);
void glLightModelfiv(Glenum pname, const Glint *params);
```

dimana :

pname : adalah spesifikasi dai model cahaya. Dapat diisi dengan salah satu nilai ini : GL_LIGHT_MODEL_AMBIENT, GL_LIGHT_MODEL_LOCAL_VIEWER , GL_ LIGHT_MODEL_TWO_SIDE.

param : untuk GL_LIGHT_MODEL_LOCAL_VIEWER , bila bernilai 0.0 artinya bahwa sudut cahaya *specular* adalah pararel terhadap z negatif dan memandang ke arah z negatif. Nilai yang lain menunjukkan bahwa pandangan adalah dari pusat. Untuk GL_LIGHT_MODEL_TWO_SIDE , bila bernilai 0.0 menunjukkan bahwa hanya bagian depan dari poligon yang diikutkan dalam perhitungan pencahayaan. Sedangkan nilai yang lain menunjukkan bahwa kedua sisi depan dan belakang diikutkan dalam perhitungan.

params: untuk GL_LIGHT_MODEL_AMBIENT atau GL_LIGHT_MODEL_LOCAL_VIEWER berarti menunjuk ke array yang berisi integer atau bilangan float. Untuk GL_LIGHT_MODEL_AMBIENT berarti menunjuk pada array yang berisi empat nilai RGBA untuk cahaya *ambient.*

3.6.4. Pengaturan Properti Material

Ada dua cara untuk mengatur properti untuk material. Cara pertama dapat dilakukan dengan menggunakan fungsi glMaterial sebelum menggambar. Contohnya dapat dilihat pada potongan program berikut ini :

```
Glfloat  gray[]={0.75f, 0.75f, 0.75f,1.0f);
...
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, gray);
```

```
glBegin(GL_TRIANGLES);
   glVertex3f(-15.0f, 0.0f, 30.0f);
   glVertex3f(0.0f, 15.0f, 30.0f);
   glVertex3f(0.0f, 0.0f, -56.0f);
glEnd();
...
```

Parameter pertama dalam `glMaterialfv` menunjukkan bagian material yang aktif. Parameter ini dapat diisi dengan `GL_FRONT`, `GL_BACK`, `GL_ FRONT_AND_ BACK`. Parameter yang kedua memberi tanda properti mana yang sedang disetting. Dalam contoh diatas properti ambient dan diffuse diberi nilai yang sama. Sedangkan parameter yang terakhir menunjukkan array RGBA dari warna. Semua primitif yang dispesifikasikan setelah pemanggilan `glMaterial` akan diperngaruhi oleh nilai ini sampai pemanggilan `glMaterial` berikutnya.

Cara kedua sering juga disebut dengan *color tracking*. Dengan *color tracking* OpenGL akan mengatur properti material dengan hanya pemanggilan `glColor`. Untuk mengenable *color tracking* dilakukan dengan memanggil fungsi `glEnable` dengan parameter *GL_COLOR_MATERIAL*. Kemudian fungsi `glColorMaterial` digunakan untuk menspesifikasikan parameter material yang diikuti dengan nilai untuk warna yang diset dengan perintah `glColor`. Contoh potongan program pada cara pertama diatas dapat ditulis dengan cara yang berbeda sebagai berikut :

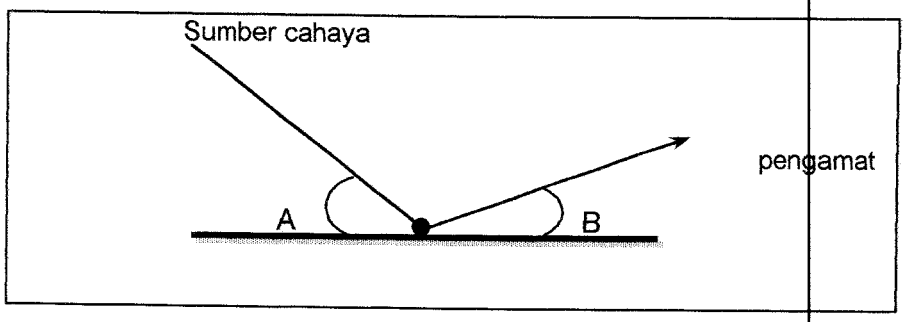
```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, gray);
```

```
...
glcolor3f(0.75f, 0.75f, 0.75f);
glBegin(GL_TRIANGLES);
   glVertex3f(-15.0f, 0.0f, 30.0f);
   glVertex3f(0.0f, 15.0f, 30.0f);
   glVertex3f(0.0f, 0.0f, -56.0f);
glEnd();
```

3.6.5. Pengunaan Sumber Cahaya

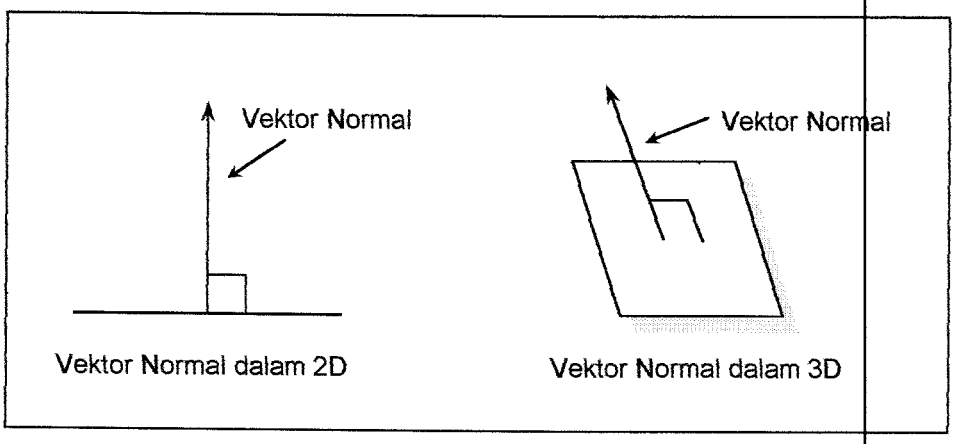
Selain memiliki nilai intensitas dan warna, sumber cahaya juga mempunyai posisi dan arah. Di dalam OpenGL dapat dibentuk delapan independen sumber cahaya yang dapat diletakkan dimana saja sesuai kebutuhan.

Ketika mengatur sumber cahaya, yang perlu dilakukan adalah menentukan letak dan arahnya. Sumber cahaya dapat bersinar ke semua arah atau hanya pada satu arah saja. Untuk semua benda yang digambar, berkas sinar dari sembarang sumber (kecuali dari sumber cahaya *ambient* murni) akan menyentuh permukaan poligon yang membentuk benda tersebut pada suatu sudut dan mengalami pemantulan. Hal ini terlihat pada gambar 3.11



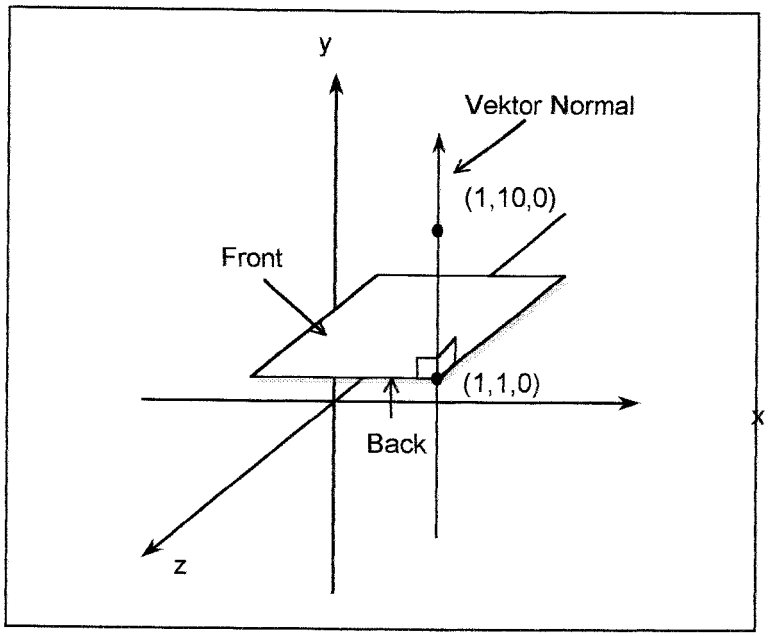
Gambar 3.11 Pemantulan berkas sinar

Pada gambar 3.11 terlihat bahwa ketika menyentuh permukaan berkas sinar membuat sudut A . Kemudian dipantulkan pada sudut B ke arah pengamat. Sudut ini adalah penghubung antara properti material dan cahaya yang telah dibahas pada bagian 3.6.1 dan 3.6.2 untuk menghitung warna yang mewakili lokasi tersebut. Untuk merealisasikan hal ini ke dalam program adalah hal yang cukup sulit karena setiap poligon terdiri dari titik-titik yang tersebar dan untuk setiap titik ini akan disentuh oleh berkas sinar. Adalah suatu hal yang tidak mungkin untuk menghitung sudut antara titik dan garis , karena terlalu banyak nilai kemungkinan untuk itu. Sebagai cara untuk memecahkan masalah tersebut untuk setiap titik harus diasosiasikan suatu informasi yang menunjukkan arah atas dan menjauhi permukaan tempat titik berada. Dalam OpenGL informasi ini berupa sebuah garis yang mempunyai titik awal pada titik yang terletak di suatu bidang kayalan (atau poligon) . Garis ini disebut vektor normal dimana vektor artinya garis berarah dan normal adalah istilah untuk menunjukkan bahwa garis tersebut tegak lurus / membuat sudut 90^0 terhadap permukaan. Gambar 3.12 berikut ini adalah contoh vektor normal dalam 2D dan 3D



Gambar 3.12 Normal vektor dalam 2D dan 3D

Penjelasan mengenai cara untuk menspesifikasikan normal dapat dilihat pada gambar 3.13 berikut ini



Gambar 3.13 Vektor normal untuk suatu permukaan

Pada gambar tersebut dapat dilihat bahwa untuk menentukan normal pada bidang dipilih titik (1,1,0) sebagai pusat garis dan titik (1,10,0) sebagai titik yang kedua. Pemilihan titik yang kedua ini juga menunjukkan bahwa arah atas adalah sumbu y positif.

Suatu vektor adalah besarnya arah yang memberi tahu OpenGL bagian mana dari poligon yang dianggap sebagai permukaan depan. Berikut ini adalah contoh potongan program yang menyeting nilai normal untuk suatu segitiga :

```
glBegin(GL_TRIANGLES);
      glNormal3f(0.0f, -1.0f, 0.0f );
      glVertex3f(0.0f, 0.0f, 60.0f);
      glVertex3f(-15.0f, 0.0f, 30.0f);
```

```
        glVertex3f(15.0f, 0.0f, 30.0f);
glEnd();
```

Fungsi `glNormal3f` mengambil 3 parameter yaitu menunjukkan arah normal dalam sumbu x,y,dan z. Namun di dalam OpengGL semua normal dari permukaan harus dikonversikan menjadi unit normal. Sebuah unit normal adalah vektor normal yang mempunyai panjang 1. Normal pada gambar 3.10 mempunyai panjang 9. Cara untuk mencari panjang dari sebuah normal dengan mengkuadratkan nilai masing-masing x, y dan z lalu tambahkan semua hasilnya lalu langkah terakhir adalah mendapatkan akar dari hasi tersebut. Setelah itu masing-masing elemen x, y, z dibagi dengan panjang normal tadi maka hasilnya berupa vektor yang mempunyai arah yang sama tetapi hanya memiliki panjang 1 unit. Tetapi semua proses ini dapat dilakukan secara otomatis oleh OpenGL dengan perintah `glEnable (GL_NORMALIZE)`.

Setelah menjelaskan semua hal yang perlu dilakukan untuk menyiapkan benda sebelum menerima cahaya maka langkah selanjutnya adalah meletakkan sumber cahaya pada posisinya. Untuk itu contoh perintah OpenGL yang digunakan adalah :

```
GLFloat ambientLight[] ={0.3f, 0.3f, 0.3f, 1.0f};
GLFloat diffuseLight[] ={0.7f, 0.7f, 0.7f, 1.0f};
...
//Melakukan setting dan enable untuk sumber cahaya ke-0
glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
//untuk menentukan posisi sumber cahaya
```

```
Glfloat LightPos[]={-50.0f, 50.0f, 100.0f, 1.0f};
glLightfv(GL_LIGHT0,GL_POSITION,LightPos);
```

Pada potongan program diatas LightPos berisi posisi dari sumber cahaya. Nilai terakhir dari array adalah 1.0 yang menunjukkan bahwa koordinat yang dimasukkan adalah posisi dari sumber cahaya. Apabila elemen terakhir ini bernilai 0.0 maka artinya bahwa sumber cahaya berada pada jarak yang sangat jauh sepanjang vektor yang ditunjukkan oleh tiga elemen sebelumnya.

BAB IV
PERANCANGAN DAN IMPLEMENTASI PERANGKAT LUNAK

Dalam bab ini dibahas mengenai perancangan dan implementasi perangkat lunak yang telah dibuat dalam tugas akhir ini, yang merupakan penerapan teori-teori yang telah diuraikan dalam bab II dan penggunaan fungsi OpenGL pada bab III. Pembahasan akan dilakukan secara garis besar dan mencakup perancangan struktur data, perancangan proses, dan rangkuman mengenai pembuatan program.

Perangkat lunak ini dikembangkan dengan bahasa pemrograman Borland Delphi versi 3.0 dan menggunakan sistem operasi Windows 95 . Penggunaan perangkat lunak ini ditambah dengan library *OpenGL for Borland* yang dikeluarkan oleh *Signsoft*. Pemanfaatan fasilitas OpenGL ini ditujukan agar fungsi-fungsi antar muka dapat disembunyikan sehingga konsentrasi pengembangan dapat difokuskan pada penerapan teori pemodelan dan visualisasi hasil.

4.1. Tujuan dan Sasaran Sistem Perangkat Lunak

Sasaran dari perangkat lunak ini adalah mampu menerima masukan data berupa informasi mengenai parameter joint-joint yang ada kemudian membentuk kerangka manusia dan memvisualisasikannya dengan menggunakan perintah-perintah OpenGL. Sedangkan tujuan perangkat lunak ini adalah untuk mengimplementasikan proses kinematika invers terhadap kerangka manusia

sehingga menghasilkan frame-frame yang berisi kedudukan atau pose kerangka manusia yang diinterpolasi untuk animasi.

4.2. Perancangan Data

Pada perangkat lunak ini menggunakan data *feature* yang ada dalam *Borland Delphi*. Kemudian data-data yang dibangun dimasukkan dalam suatu kelas.

Secara garis besar data-data yang digunakan dalam perangkat lunak ini dibedakan menjadi tiga jenis, yaitu : data masukan , data yang diproses dan data keluaran.

4.2.1 Data Masukan

Data joint yang digunakan perangkat lunak ini adalah sebuah berkas teks yang berisi diskripsi mengenai karakteristik atau parameter seluruh joint yang membentuk kerangka manusia. Untuk setiap joint yang ada akan memiliki data-data seperti yang tertera pada tabel 4.1 berikut ini :

Tabel 4.1 Isi Data Joint

	Nama	Jenis	Panjang	Keterangan
1	Nama joint	Karakter	10	Menunjukkan nama joint
2	child	byte	1	merupakan data yang menunjukkan child dari joint
3	parent	byte	1	merupakan data yang menunjukkan parent dari joint
4	parameter θ	GLFloat	6	merupakan data parameter θ
5	parameter α	GLFloat	6	merupakan data parameter α
6	parameter d	GLFloat	6	merupakan data parameter d
7	parameter a	GLFloat	6	merupakan data parameter a
8	RotX	byte	1	keterangan mengenai perputaran pada sumbu x

9	RotY	byte	1	keterangan mengenai perputaran pada sumbu y
10	RotZ	byte	1	keterangan mengenai perputaran pada sumbu z
11	LowerB	GLFloat	6	Batas bawah untuk nilai parameter θ
12	UpperB	GLFloat	6	Batas atas untuk nilai parameter θ

GLFloat adalah tipe data float yang disediakan oleh OpenGL sebagai tambahan tipe data. Berikut adalah tata berkas yang digunakan untuk membentuk susunan joint pada manusia.

```
[num joint]
<Nama><Child><Parent><θ><α><d><a><RotAtX><RotAtY><RotAtZ>...
.....<LowerBound><UpperBound>
```

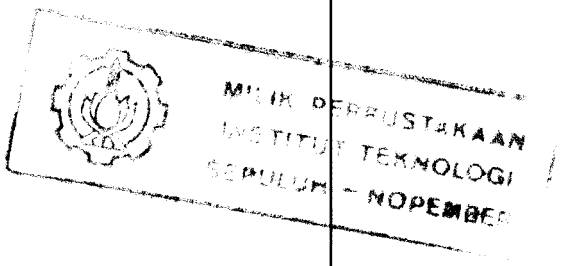
Posisi *end-effector* lama adalah posisi end effector yang akan digerakkan dan posisi *end-effector* baru adalah posisi tujuannya yang baru. Posisi parent adalah posisi parent joint yang dipilih sebagai acuan pergerakan *end-effector*

4.2.2 Data Utama Pada Saat Proses

Pada saat perangkat lunak bekerja memodelkan dan melakukan interaksi masukan seperti yang terdapat pada data masukan akan menghasilkan data-data berikut :

1. Tenaga tarikan, merupakan data hasil perhitungan proses tenaga tarikan yang mendapat masukan berupa posisi end-effector lama dan posisi end-effector baru.

2. Matrik Jacobian berupa matrik yang setiap kolomnya berisi hasil persamaan jacobian.
3. Matrik theta adalah hasil perkalian antara tenaga tarikan dan matrik jacobian. Data ini kemudian di proses integrasi.
4. Data integral matrik theta merupakan hasil integrasi dan akan digunakan dalam proses kinematika maju untuk mendapatkan posisi *end-effector* yang baru
5. Data joint dalam Koordinat windows, data yang berisikan koordinat pada windows. Semua data koordinat joint akan diubah dalam proses transformasi menjadi koordinat OpenGL yang diperlukan untuk melakukan proses penggambaran.
6. Data cahaya dan permukaan, data yang berisi informasi mengenai permukaan dan properti dari cahaya untuk permukaan tersebut. Data ini merupakan hasil dari proses pencahayaan dari fungsi OpenGL.
7. Data cahaya, material dan permukaan, data yang berisi data permukaan berikut dengan data properti dari cahaya dan material untuk permukaan tersebut. Data ini dihasilkan dari proses pencahayaan dan proses inisialisasi material yang menggunakan *library* OpenGL.
8. Koordinat mata, koordinat yang sesuai dengan pandangan mata manusia
9. Koordinat *Clipped*, adalah koordinat hasil dari proses transformasi proyeksi dimana semua data yang berada diluar *viewing volume* yang digunakan tidak akan digambar.
10. Koordinat *Normalized Device*, adalah koordinat clipped dibagi dengan koordinat w sebagai hasil dari proses *Perspective Division*



11. Koordinat *Window*, koordinat akhir dari proses transformasi yang digunakan OpenGL untuk menggambar obyek pada layar

4.2.3 Data Keluaran

Hasil pengolahan dari perangkat lunak ini adalah data frame yang menampilan secara sequensial pose-pose yang ada atau dihasilkan. Data frame ini merupakan berkas teks dan berisi parameter θ dari setiap joint dan posisi joint tersebut dalam ruang seperti tercantum dalam tabel 4.2. Besarnya data ini tergantung pada jumlah frame yang ada. Data inilah yang nantinya mengalami proses interpolasi dan hasil interpolasinya merupakan animasi bentuk gerakan manusia.

Tabel 4.2 Isi Data Frame

	Nama	Jenis	Panjang	Keterangan
1	Parameter θ	Karakter	10	Menunjukkan nama joint
2	Posisi X	GLFloat	1	menunjukkan posisi X dari joint dengan parameter θ
3	Posisi Y	GLFloat	6	menunjukkan posisi Y dari joint dengan parameter θ
4	Posisi Z	GLFloat	6	menunjukkan posisi Z dari joint dengan parameter θ

4.3. Perancangan Proses

Perangkat lunak ini terdiri atas dua proses utama yaitu proses perhitungan kinematika invers hasil interaksi dengan pengguna pada kerangka dan proses

visualisasi data. Secara garis besar kedua proses tersebut akan dibahas pada baian berikut ini.

4.3.1 Perancangan Proses Perhitungan Kinematika Invers

Proses perhitungan kinematika invers adalah proses yang menghitung posisi *end-effector* menuju titik yang diinputkan oleh pengguna dihitung dari *base*. Hasilnya adalah data sudut-sudut θ dari joint yang ke-*base* hingga *end-effector* dan posisi joint-joint tersebut pada ruang 3D. Masukan yang dipakai adalah dari pengguna yaitu dengan interaksi langsung menginputkan dari hasil *click mouse*. Selanjutnya posisi baru ini digunakan dalam proses perhitungan hingga posisi end-effector sampai atau mendekati posisi yang diinginkan.

Proses perhitungan kinematika invers meliputi proses perhitungan tenaga tarikan, perhitungan matrik jacobian , perhitungan skala, proses perkalian matrik jacobian dengan tenaga tarikan dan integrasi.

4.3.2 Perancangan Proses visualisasi

Proses visualisasi menggunakan *library* OpenGL . Langkah-langkahnya dibagi menjadi empat bagian yaitu transformasi dari titik 3D menjadi koordinat OpenGL, pencahayaan pada obyek , penentuan properti material, dan proses penggambaran obyek pada layar.

Proses transformasi bertujuan merubah koordinat joint yang berupa koordinat ruang 3D menjadi koordinat OpenGL. Proses transformasi ini terdiri dari empat proses sesuai dengan penjelasan pada Bab III gambar 3.5.

Proses penggambaran obyek kerangka manusia merupakan penggambaran titik-titik posisi joint sesuai dengan perhitungan posisi terhadap θ yang dimiliki. kemudian dilanjutkan dengan penggambaran *link*-nya.

Untuk proses penggambaran link, terlebih dahulu diolah dengan proses pencahayaan dan pengaturan properti material untuk benda yang mewakili *link* tersebut sehingga gambar yang dihasilkan lebih menunjukan efek tiga dimensi yang sesungguhnya.

Proses pencahayaan adalah proses perhitungan cahaya untuk permukaan dan elemen sumber cahaya apa saja yang akan digunakan. Data permukaan benda tadi dimasukkan ke dalam proses pencahayaan menghasilkan data permukaan benda yang sudah mengandung elemen cahaya untuk selanjutnya digunakan dalam proses pengaturan properti material.

Pada proses pengaturan properti material ini permukaan benda mendapat 4 elemen material. Untuk keterangan lebih lanjut baca Bab III bagian Pengaturan cahaya pada ruang dan Pengaturan Properti Material. Untuk Selanjutnya obyek yang digambar akan memiliki data cahaya dan material sesuai yang diinginkan.

4.4. Perancangan Antarmuka

Untuk menampilkan perangkat lunak hasil implementasi dari bagian maka dibuat sebuah window utama yang di dalamnya terdapat 2 macam menu. Yaitu menu samping dan menu toolbar. Menu samping berguna untuk memanipulasi obyek manusia mulai dari pemberian warna, penentuan posisi cahaya, control gerakan manusia dengan metode kinematika maju dan kinematika invers serta

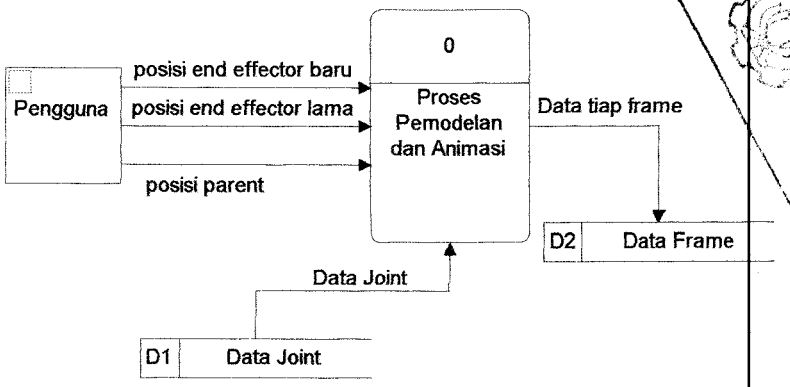
manipulasi frame animasi. Sedangkan menu toolbar berisi fasilitas-fasilitas perangkat lunak pada umumnya yaitu : new, open, save, print dan exit ditambah dengan menu tambahan berupa control tampilan window yang berupa tampilan obyek manusia, background gambar, jenis proyeksi, zoom in, zoom out, perintah untuk menampilkan hasil animasi dan juga help. Pemilihan dua sub menu yang berbeda ini untuk menghindari kompleksitas menu pada toolbar dan memperkecil penggunaan form tambahan dan mempermudah pengguna dalam proses penentuan pose-pose frame untuk animasi.

Untuk menu samping bagian pewarnaan disediakan pilihan untuk menentukan warna material, warna cahaya yang datang dan posisinya, serta pilihan untuk menampilkan obyek manusia secara transparan. Pada bagian Kinematika maju terdapat button *setjoint position* untuk memilih joint yang akan digerakkan dan up-down counter pada sudut theta untuk mengontrol pergerakkannya. Menu ini terdapat juga informasi joint yang sedang diselect. Untuk menu kinematika invers juga terdapat button *setjoint position* untuk memilih joint yang akan digerakkan dan pilihan parent joint serta informasi posisi akhir *end-effector*. Menu samping yang terakhir adalah Frame yang didalamnya terdapat pilihan untuk menambah, mengurangi jumlah frame.

4.5. Diagram Alir Data

Diagram Alir Data (DAD) untuk perangkat lunak ini ditunjukkan dalam gambar-gambar berikut ini. Gambar 4.1 menampilkan Diagram alir data level 0 dari aplikasi animasi kinematik gerakan manusia.

Data yang akan digunakan pada DAD level 0 ini berasal dari user baik masukan langsung saat berinteraksi dengan perangkat lunak ataupun berasal dari berkas masukan yang bertipe teks. Untuk masukan langsung, berupa joint mana yang akan digerakkan, masukkan parent dari joint yang digerakkan, serta posisi koordinat *end-effector.* Untuk masukan dari berkas teks berupa data informasi parameter joint. Data frame merupakan hasil keluaran dan disimpan dalam berkas teks yang nantinya dapat dibuka kembali jika ingin ditampilkan atau melakukan perubahan.

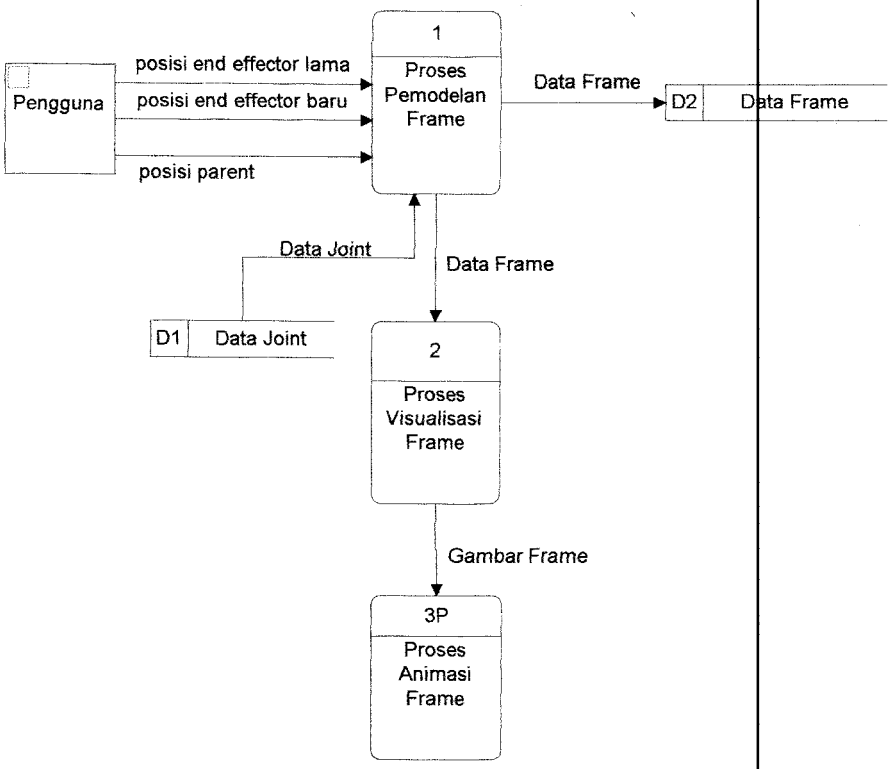


Gambar 4.1 DAD level 0 , Proses pemodelan dan Animasi Frame

Diagram alir data level 0 diatas memiliki level 1 seperti yang ditunjukkan dalam Gambar 4.2. yang masing-masing prosesnya akan diuraikan lebih lanjut menjadi level 2 seperti yang ditampilkan dalam Gambar 4.3. dan 4.4.

Data pada DAD level 1 yaitu Posisi end-effector lama, posisi end-effector baru, dan posisi parent dipakai dalam proses perhitungan kinematika invers untuk menghasilkan data frame dan data frame selanjutnya digunakan untuk proses

visualisasi (penggambaran) pada layar. Hasil gambar frame ini pada proses terakhir akan dianimasikan.

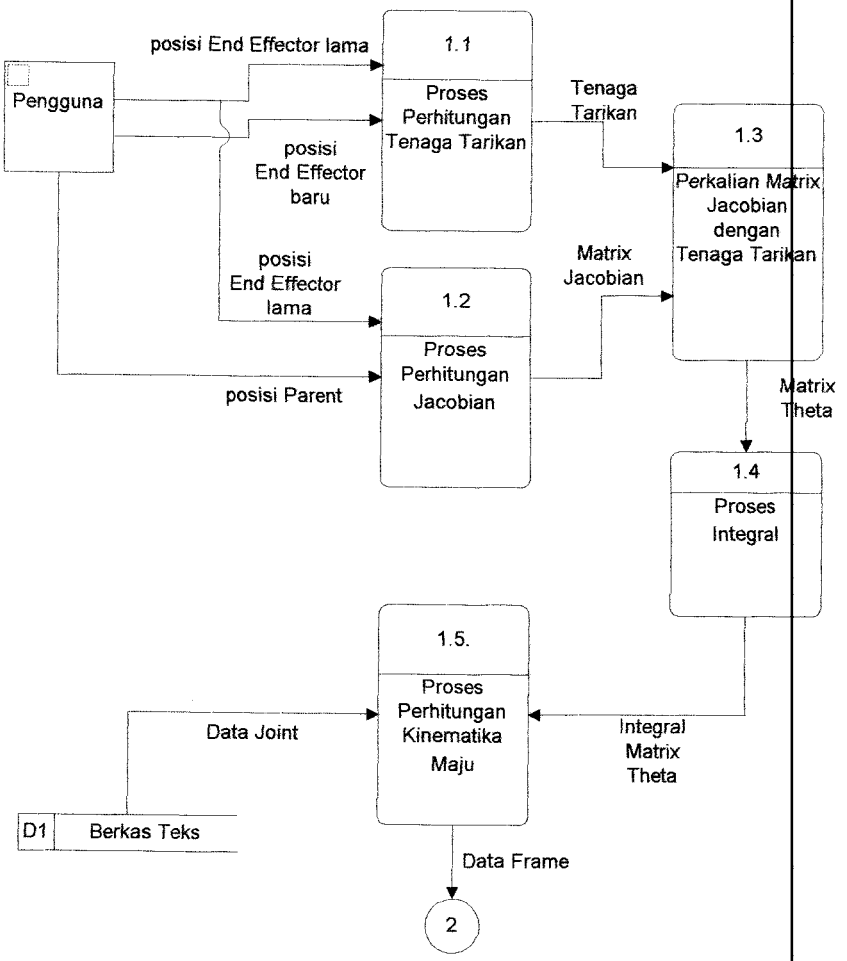


Gambar 4.2 DAD level 1 , Proses pemodelan , Visualisasi dan Animasi Frame

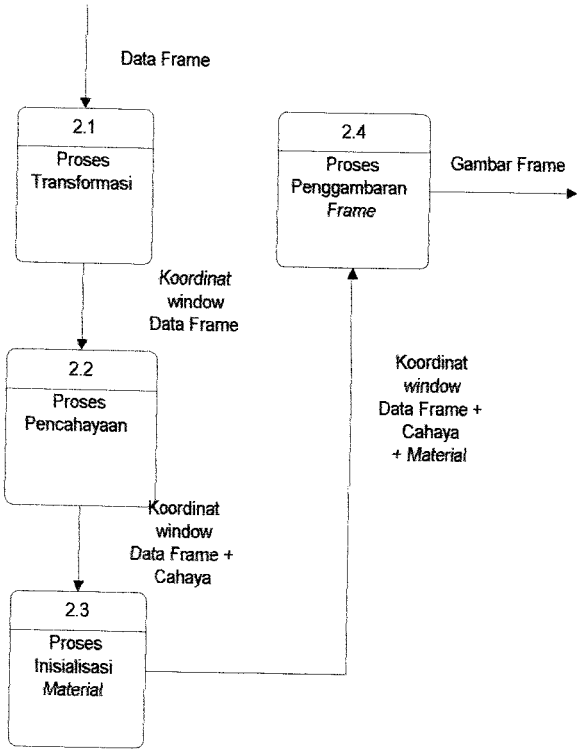
Pada DAD level 2 proses perhitungan kinematika invers posisi end-effector lama dan baru digunakan untuk menghitung tenaga tarikan yang terjadi. Sedangkan untuk mendapatkan matrik jacobian digunakan data posisi end-effector lama dan posisi parent.

Proses perkalian tenaga tarikan dan matrik jacobian akan menghasilkan matrik theta yang merupakan parameter theta untuk semua joint yang terletak antara joint parent dan joint end-effector. Sebelum digunakan dalam proses perhitungan

kinematika maju maka matrik ini mengalami proses integrasi terlebih dahulu. Hasil dari keseluruhan proses berupa data frame yang berisi keseluruhan posisi joint-joint yang ada dan thetanya.



Gambar 4.3 DAD level 2 , Proses Perhitungan Kinematika Invers

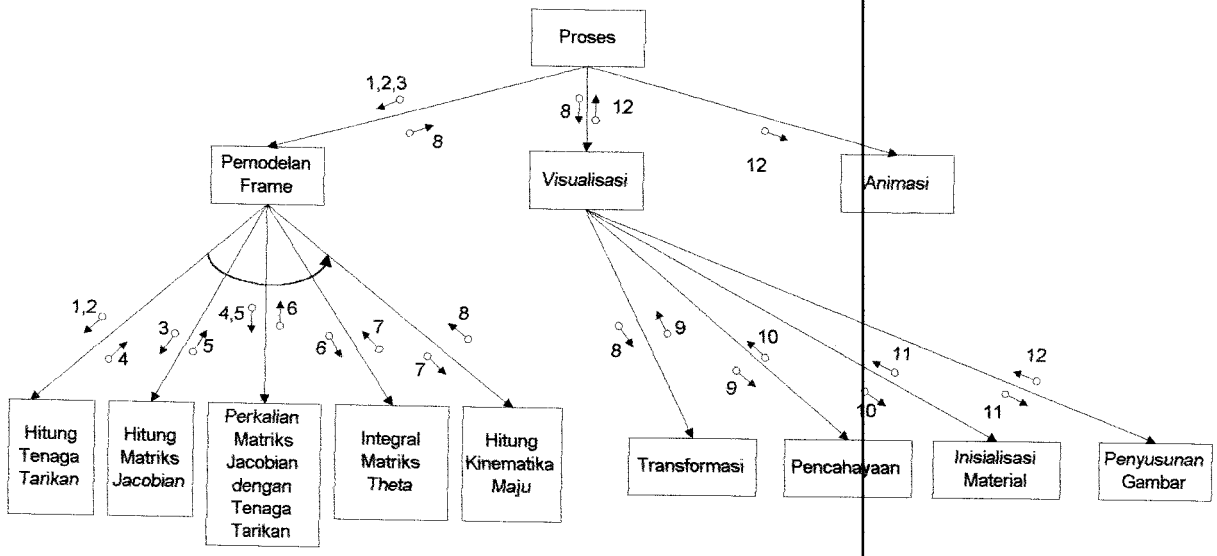


Gambar 4.4 DAD level 2 , Proses Visualisasi Frame

Gambar 4.4. menunjukkan Diagram alir data yang terjadi dalam proses transformasi yang merubah koordinat vertex menjadi koordinat *windows*

4.6. Hirarki Modul

Diagram disain modul yang diimplementasikan pada Tugas Akhir ini secara keseluruhan dapat dijabarkan seperti pada Gambar 4.5. Nomor ini menunjukkan aliran data yang masuk dan keluar dari proses dan dijelaskan sebagai berikut :



Gambar 4.5. Hirarki modul utama dari Animasi Kinematik Gerakan Manusia

1. Posisi end-effector lama
2. Posisi end-effector baru
3. Posisi parent
4. Tenaga tarikan
5. Matriks Jacobian
6. Matriks Theta
7. Integral Matriks Theta
8. Posisi end effector akhir (data frame)
9. Koordinat windows data frame
10. Koordinat windows data frame+data cahaya
11. Koordinat windows data frame+data cahaya+data material
12. Gambar frame

Proses pemodelan frame merupakan proses iterasi yang menghitung posisi joint dari based hingga effektor yang mengalami perubahan akibat adanya tenaga tarikan. Tenaga tarikan ini digunakan untuk perhitungan yang akan meletakkan *end-effector* pada suatu posisi. Proses pemodelan melalui tahap-tahap perhitungan tenaga tarikan, matrik jacobian, perkalian antara hasil dari keduanya, integrasi terhadap matrik theta dan perhitungan kinematika maju . Proses iterasi ini dilakukan hingga posisi akhir *end- effector* berada pada tempat yang diinginkan atau hingga semua joint telah mencapai nilai theta maksimumnya.

4.7. Implementasi Struktur Data

Untuk menunjang perancangan perangkat lunak diciptakan beberapa tipe data utama meliputi TLightColor, TLookAt, TPoints, TData dan kelas data utama meliputi kelas TSceneGL, Tkinematic, TInterpolate dan Tobject . Dan terdapat kelas kelas tambahan seperti Tfile dan Tmatrik untuk membantu perhitungan dan manipulasi file .

4.7.1 Tipe Data

Untuk Mendefinisikan dan menyimpan warna obyek baik untuk material maupun pencahayaan diciptakan tipe array TLightColor yang terdiri dari empat elemen yaitu warna merah, hijau, biru, dan keterangan tambahan yang bertipe GLfloat. Tipe ini berguna untuk mendefinisikan variabel *ambient*, *diffuse* dan *specular* baik untuk pencahayaan maupun properti material. Untuk pandangan yang terdiri dari empat macam pandangan yaitu dari tampak depan, samping kanan , atas

dan perspektif disimpan dalam array TLookAt yang berisi sembilan nilai yang diperlukan untuk meletakkan kamera / pengamat seperti yang telah dijelaskan dalam bab III.

```
type
TLookAt      = array[1..9] of GlFloat;
TLightColor = array[0..3] of GlFloat;

fLight0Ambient,fLight0Diffuse, fLight0Specular,
fLight0Pos                                          : TLightColor;
fMatAmbient, fMatDiffuse, fMatSpecular                    : TLightColor;

const
 PerspLookAt: TLookAt = (  16.0, -16.0, 16.0,
                           0.0,  0.0, 0.0,
                           1.0,  0.0, 0.0);
 FrontLookAt: TLookAt = (  0.0,-12.0,0.0,
                           0.0,0.0,0.0,
                           1.0,0.0,0.0);
 TopLookAt  : TLookAt = (  12.0,0.0,0.0,
                           0.0,0.0,0.0,
                           0.0,1.0,0.0);
 RightLookAt: TLookAt = (  0.0,0.0,12.0,
                           0.0,0.0,0.0,
                           1.0,0.0,0.0);
```

Untuk menyimpan data koordinat (x,y,z) diciptakan tipe record TPoints yang terdiri dari variabel x,y,z bertipe GLfloat. Sedangkan data lengkap dari joint manusia dalam class TData. Semua data pada saat pemrosesan disimpan dalam class ini, untuk selanjutnya dipakai dalam proses visualisasi dan animasi obyek manusia. Kecuali untuk data joint dalam koordinat mata, dalam koordinat clipped, dalam koordinat normalized device dan dalam koordinat windows semuanya sudah di handle oleh fungsi-fungsi transformasi OpenGL.

```
const MaxFrame = 30;
      MaxJoint = 30;

type
  TPoints = record
              x,y,z : GLFloat;
            end;

  TJoint    = record
                Names     : string[20];
                Child,parent:byte;
                Theta,
                a,
                Alpha,
                d               : glFloat;
                LBound,Ubound : glFloat;
                stop            : boolean;
                Miror           : Boolean;
                Position : TPoints;

                RotAt     : TPoints;
              end;
  ArrJoint = array[1..MaxJoint] of TJoint;
  ArrPoints =array[1..MaxJoint] of TPoints;
  PArrPoints =^ArrPoints;

  Tq      = record
              q    : array[1..MaxJoint] of GlFloat;
              Position : array[1..MaxJoint] of TPoints;
            end;
  ArrQ    = array[1..MaxFrame] of TQ;
  PArrQ   = ^ArrQ;

  TData = class(Tobject)
    Points      : PArrPoints;
    TotFrame    : Integer ;
    Frame       : PArrQ;
    nJoint      : byte;
    Joint       : ArrJoint;
  end;
```

4.7.2 Kelas TSceneGL

Dalam kelas ini terdapat data-data dan fungsi-fungsi yang berhubungan dengan persiapan visualisasi obyek menggunakan OpenGL. Kelas ini yang mengatur interaksi pemakai dengan perangkat lunak . Di dalamnya termasuk proses

transformasi *viewing, modelling, projection*, *viewport*, serta inisialisasi material dan proses pencahayaan.

```
TsceneGL=class(Tobject)
    Fperspective            : boolean;
    Angle,DistNear,DistFar  : single;
    Range                   : GlFloat;
    fLight0Ambient,fLight0Diffuse, fLight0Specular,
    fLight0Pos : TLightColor;
    fMatAmbient, fMatDiffuse, fMatSpecular: TLightColor;
    fShine                  : glfloat;
    Constructor create;
    Procedure Projection(width,height:integer;L:TLookAt);
    Procedure Modelling;
    function  GetCoordinateOGL(X,Y:Integer):TPoints;
    function  FindJointOnClick(X,Y:integer;views:byte;
                var nPiF:integer;nPi:integer;J:TQ):boolean;
    procedure DrawHuman(nFrame:byte;D:TData);
    Procedure DrawFrame(nFrame:byte;D : TData);
    Procedure Lighting_and_Material(Blend:Boolean);
    Procedure InitLight;
    Procedure RotateX(Angle : GLFloat;nFrame:byte;var D :TData);
    Procedure RotateY(Angle : GLFloat;nFrame:byte;var D :TData);
    Procedure RotateZ(Angle : GLFloat;nFrame:byte;var D :TData);
    Procedure Translate(X,Y,Z:GLFloat;nFrame:byte;var D :TData);
    Procedure DrawGround;
  end;
```

Dalam kelas ini juga terdapat prosedur mencari joint yang dipilih user melalui interaksi langsung dengan mouse dan juga prosedur menggerakkan joint tersebut. Tiap posisi joint yang dipilih masih berupa koordinat windows dan harus ditransformasi menjadi koordinat OpenGL data Joint mana yang dipilih untuk digerakkan.

4.7.3 Kelas TKinematic

Dalam kelas ini terdapat data-data dan fungsi-fungsi yang berhubungan dengan perhitungan yang diperlukan untuk proses kinematika invers. Penjelasan mengenai proses yang dilakukan dapat dilihat pada penjelasan bab II.

```
TKinematic = class(Tobject)
private
 P0        : MatrixType;
 Function  Integral  Based,EndEff,NFrame:byte;q: MatrixType;
                     var D:TData):MatrixType;
 Function  ConstructJacobian(root,Based,EndEff,NFrame:byte;
                              D:TData):MatrixType;
 Function  FindAxis  root,i     ,NFrame:byte;
                     D:TData):TPoints;
 Function  CekDist (DA,DB:TPoints;v:byte):Boolean;
 Function  CekTheta(Based,EndEff,NFrame:byte;D:TData):Boolean;
 Function  ComputeForce(Xdes,XCur:TPoints;View:byte):TPoints;
 Function  Input(Theta,alpha,a,d:glfloat):MatrixType;
 Function  Inverse_Input(Theta,alpha,a,d:glfloat):MatrixType;
 Function  ForwardKinematics (Root,Jointi,NFrame:byte;D:TData)
           :MatrixType;
 Procedure SetJointPosition(Root,Based,EndEff,NFrame:byte;
                            Var D:TData);
 Function  ComparePosLegX(Left,Right,View:byte;D:TQ):glFloat;
 Function  Translate(X,Y,Z:GLFloat):MatrixType;
public
 Floorx    : glFloat;
 FloorxA   : glFloat;
 PosPel,PosPelA:TPoints;
 constructor create;
 Procedure SetAllJointPosition  (NFrame:byte;Var D:TData);
 Function JacobianTransposeMethods(root,Based,EndEff,NFrame:byte;
                          Xd:TPoints;var D:TData;view:byte):boolean;
 Procedure JointTranslate(X,Y,Z:glFloat;NFrame:byte;Var D:TData);
end;
```

4.7.4 Kelas TInterpolate

Dalam kelas ini terdapat data-data dan fungsi-fungsi yang berhubungan dengan perhitungan yang diperlukan untuk proses Interpolasi hasil inputan frame pada saat interaksi dengan user. Jumlah Frame minimum adalah satu dan maksimumnya adalah 30. Penjelasan mengenai proses yang dilakukan dapat dilihat pada penjelasan bab II.

```
TInterpolate = class(TObject)
    DP,CP,IP    : Points2D;
    UKnots      : array[0..40] of GlFloat;
    Fr          : array[1..6,1..18] of TPoints;F          : Byte;
    procedure DrawNURBSCurve(n,p:integer; Uknots :array of glfloat;
       ControlPoints:Points2D; var InterpolatePoint : Points2D;
        Var Index:byte);
    procedure Computeuk(n:integer; DP:Points2D;
                        var uknot:array of float);
    procedure ComputeU(n,p:integer; uknot:array of float;
                        var UKnots:array of float);
    procedure Basis(i:integer; u:glfloat; p:integer;
                  knots:array of glfloat; var N : array of glfloat);
    function FindSpan(n,p: integer; knot:float;
                  Knots:array of float):integer;
    procedure GlobalCurveInterp(n:integer; DP:Points2D; p: integer;
                  var U:array of float; var CP:Points2D);
  end;
```

4.7.5 Kelas Tfile

Kelas Tfile merupakan kelas tambahan yang berisi prosedur dan fungsi-fungsi yang diperlukan untuk pembacaan file data joint, pembacaan berkas file yang sudah ada dan penyimpanan hasil .

```
TFile = class(Tobject)
    Procedure ReadFromFile(Filename:String;D:TData);
    Procedure JointFromFile(Filename:String;D:TData);
    Procedure SavesToFile(Filename:String;D:TData);
    Function  GetToken(const s:string;var cp:word):string;
    Function  StrToGLFloat(s:string):GlFloat;
  end; {TReadFile}
```

4.7.6 Kelas TMatrik

Kelas Tmatrik juga merupakan kelas tambahan yang berisi prosedur dan fungsi-fungsi yang diperlukan untuk proses perkalian, pengurangan matrik dan

vektor. Prosedur dan fungsi ini penting dalam perhitungan kinematika dalam kelas Tkinematic.

```
MatrixType     = array[1..MaxJoint,1..MaxJoint] of glFloat;
TMatrix = class(Tobject)
    Function  MultiplyMatrix(M1,M2:MatrixType;
                  RowM1,col,rows:byte):MatrixType;
    Function  AddMatrix(M1:MatrixType;M2:MatrixType;
                  col,rows:byte):MatrixType;
    Function  SubtractMatrix(M1:MatrixType;M2:MatrixType;
                  col,rows:byte):MatrixType;
    Function  Transpose(M:MatrixType;nCol:byte):MatrixType;
    Procedure IdentityMatrix(var m:MatrixType;n:byte);
    Function  VectorCrossProduct(u,v:TPoints):TPoints;
    Function  VectorSubtract(u,v:TPoints):TPoints;
  end;
```

4.7.7 Kelas TObject

Kelas TObject merupakan kelas abstrak yang disediakan oleh Delphi dan jika akan membuat kelas baru maka kelas tersebut dapat dianggap sebagaiturunan dari TObject sehingga untuk masing-masing obyek permukaan dibuat beberapa kelas yang merupakan turunan dari Tobject.

4.8. Implementasi Proses

Proses utama perhitungan Kinematika Invers meliputi perhitungan terhadap tenaga tarikan yang diberikan kepada *end-effector*, perhitungan untuk matrik Jacobian untuk konfigurasi saat itu dan menghitung besarnya *joint velocities*. Hasilnya akan diintegrasi sebayak satu kali untuk mendapatkan konfigurasi baru untuk *end-effector*.

Untuk menghitung tenaga tarikan dibutuhkan posisi *end-effector* baru dan posisi *end-effector* lama. Sebelum menghitung tenaga tarikan ini perlu dihitung terlebih dahulu posisi *end-effector* lama. Hal ini perlu mengingat proses kinematika invers adalah proses iterasi, sehingga untuk setiap iterasi maka posisi *end-effector* lama akan terus berubah sesuai perubahan nilai sudut jointnya.

```
function  TKinematic.ComputeForce;
var Temp      : Tpoints;
begin
  case view of
  1: begin
       temp.x:=Xdes.x-Xcur.x;
       temp.y:=Xdes.y-Xcur.y;
       temp.z:=0;
     end;
  2: begin
       temp.x:=0;
       temp.y:=Xdes.y-Xcur.y;
       temp.z:=Xdes.z-Xcur.z;
     end;
  3: begin
       temp.x:=Xdes.x-Xcur.x;
       temp.y:=0;
       temp.z:=Xdes.z-Xcur.z;
     end;
  end;
  result:=temp;
end;
```

Proses selanjutnya adalah perhitungan untuk mendapatkan matrik jacobian yang menunjukkan sudut perubahan joint dari parent sampai *end-effector*. Proses perhitungan matrik jacobian ini dibagi lagi menjadi proses pencarian axis, pengurangan vektor axis joint dengan posisi joint dan vektor hasilnya dikalikan dengan axis.

```
Function  TKinematic.ConstructJacobian;
var J,JT          : MatrixType;
    i             : byte;
    t1,t2,axis    : TPoints;
begin
  for i:=based to EndEff do
  begin
    axis:=FindAxis(Root,i,NFrame,D);
    t1:=Mat.VectorSubtract(D.Frame[NFrame].Position[EndEff],
                           D.Frame[NFrame].Position[i]);
    t2:=Mat.VectorCrossProduct(t1,axis);

    J[i-based+1,1]:=t2.x;
    J[i-based+1,2]:=t2.y;
    J[i-based+1,3]:=t2.z;
    J[i-based+1,4]:=axis.x;
    J[i-based+1,5]:=axis.y;
    J[i-based+1,6]:=axis.z;
  end;
  JT:=Mat.Transpose(J,EndEff-Based+1);
  result:=J;
end;
```

Setelah mendapatkan matrik jacobian maka selanjutnya matrik hasil ini akan dikalikan dengan tenaga tarikan untuk menghasilkan vektor sudut joint. Setelah itu proses dilanjutkan dengan integrasi vektor sudut joint ini untuk mendapatkan sudut-sudut joint dari parent hingga *end-effector*

```
Function  TKinematic.Integral;
var i : byte;
    h : glFloat;
    temp: glFloat;
begin
  for i:=Based to EndEff do
  begin
    if D.Joint[i].Miror then  h:=-0.5 else h:=0.5;
    temp:=(h*q[i-based+1,1]);
    if (not D.Joint[i].stop) then
    begin
      if (( (D.Frame[NFrame].q[i]+temp)>=D.Joint[i].UBound) or
          ( (D.Frame[NFrame].q[i]+temp)<=D.Joint[i].LBound))then
      begin
        if ( (D.Frame[NFrame].q[i]+temp)>=D.Joint[i].UBound)  then
          D.Frame[NFrame].q[i]:=D.Joint[i].UBound;
```

```
        if ( (D.Frame[NFrame].q[i]+temp)<=D.Joint[i].LBound)  then
          D.Frame[NFrame].q[i]:=D.Joint[i].LBound;
        D.Joint[i].stop:=true
      end
      else
        D.Frame[NFrame].q[i]:=D.Frame[NFrame].q[i]+temp;
    end;
  end;
end;
```

Setelah mendapatkan sudut joint maka hasil ini dimasukkan dalam prosedur *SetJointposition* untuk mendapatkan posisi joint *end-effector* yang baru. Cara untuk mendapatkan posisi ini dilakukan dengan prosedur kinematika maju.

```
Procedure TKinematic.SetJointPosition;
var  a          : byte;
     T,Pi       : MatrixType;
begin
  P0[1,1]:=0;
  P0[2,1]:=0;
  P0[3,1]:=0;
  P0[4,1]:=1.0;
  D.Frame[NFrame].Position[root].x:=0;
  D.Frame[NFrame].Position[root].y:=0;
  D.Frame[NFrame].Position[root].z:=0;
  for a:=Based to EndEff-1 do
  begin
    T:=ForwardKinematics(Root,a,NFrame,D);
    Pi:=Mat.MultiplyMatrix(T,P0,4,4,1);
    D.Frame[NFrame].Position[a+1].x:=Pi[1,1];
    D.Frame[NFrame].Position[a+1].y:=Pi[2,1];
    D.Frame[NFrame].Position[a+1].z:=Pi[3,1];
  end;
end;
```

Iterasi diatas dilakukan hingga posisi joint mencapai target . Tetapi sebelumnya perlu dilakukan pengecekan apakah semua sudut joint telah mencapai batas maksimum/minimumnya. Ini untuk mengetahui apakah terdapat pemecahan atau tidak.

Proses Visualisasi secara garis besar terdiri dari terdiri dari proses transformasi *viewing, modelling, viewport* dan *projection*. Proses transformasi *viewing* dan *modelling* dijabarkan dalam prosedur TSceneGL.Projection dan prosedur TsceneGL.Modelling dibawah ini

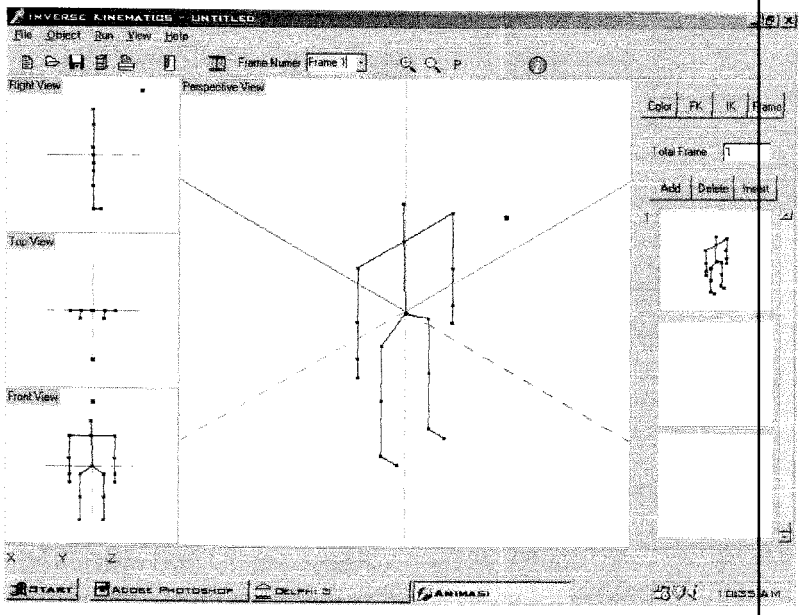
```
Procedure TsceneGL.Projection;
var
  gldAspect : GLdouble;
  ratio:GlFloat;

begin
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity;
  If Fperspective then
  begin
    gldAspect := width/height;
    gluPerspective(Angle, gldAspect, DistNear, DistFar);

  end
  else
  begin { Orthogonal projection}
    If width<=height then
    begin
      ratio:=height/width;
        GlOrtho(-range,range,-range*ratio,range*ratio,1,100);
    end
      else
    begin
      ratio:=width/height;
      GlOrtho(-range*ratio,range*ratio,-range,range,1,100);
    end;
  end;
  gluLookAt(L[1],L[2],L[3],L[4],L[5],L[6],L[7],L[8],L[9]);
  glViewport(0, 0, width, height)
end;

Procedure TsceneGL.Modelling;
begin
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity;
end;
```

Sedangkan proses pencahayaan dan properti material obyek manusia dapat dijabarkan dalam prosedur TSceneGL.Lighting_and_Material dibawah ini yang telah dijelaskan pada bab III .

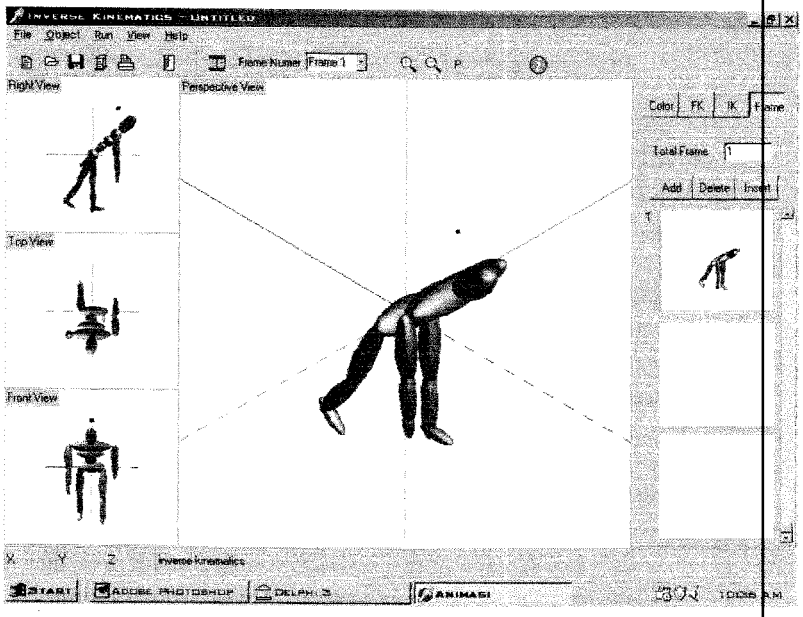
```
Procedure TSceneGL.Lighting_and_Material;
begin
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glLightfv(GL_LIGHT0,GL_AMBIENT,@fLight0Ambient);
   glLightfv(GL_LIGHT0,GL_DIFFUSE,@fLight0Diffuse);
   glLightfv(GL_LIGHT0,GL_SPECULAR,@fLight0Specular);
   glLightfv(GL_LIGHT0,GL_POSITION,@fLight0Pos);

   glEnable(GL_DEPTH_TEST);
   glDepthFunc(GL_LEQUAL);
   glEnable(GL_AUTO_NORMAL);
   glEnable(GL_NORMALIZE);

   if Blend then
   begin
     glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
     glEnable(GL_BLEND);
   end;
   glMaterialfv(GL_FRONT,GL_AMBIENT,@fMatAmbient);
   glMaterialfv(GL_FRONT,GL_DIFFUSE,@fMatDiffuse);
   glMaterialfv(GL_FRONT,GL_SPECULAR,@fMatSpecular);
   glMaterialf(GL_FRONT,GL_SHININESS,fShine);
end;
```

4.9. Implementasi Antar muka

Hasil dari perancangan antarmuka dapat dilihat pada gambar 4.6 dan pada gambar 4.7 menunjukkan hasil suatu frame setelah mendapat interaksi dari pengguna.



Gambar 4.6. Tampilan Awal Antarmuka



Gambar 4.7. Hasil Interaksi dengan Pengguna

BAB V

HASIL UJI COBA

PERANGKAT LUNAK

Dalam bab ini dijelaskan mengenai hasil uji coba pada perangkat lunak yang diimplementasikan dalam Tugas Akhir ini, dengan menggunakan spesifikasi perangkat keras sebagai berikut :

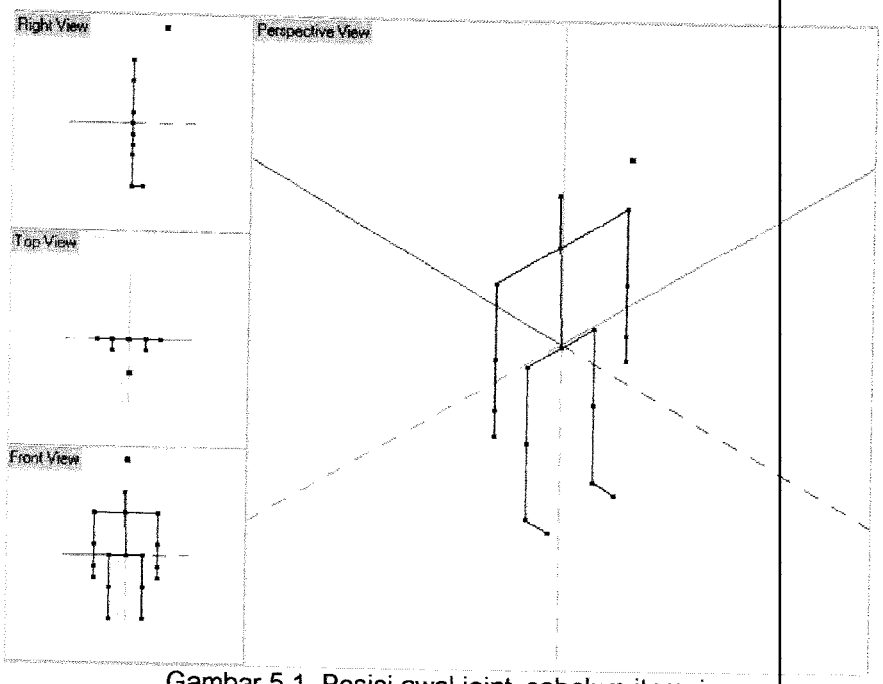
- Prosesor Intel Pentium 133
- Memory Fisik 32 Megabyte EDO
- VGA Memory 2 Megabyte
- Display 65.535 warna
- Hard Disk 2,1 Megabyte

Sedangkan kebutuhan perangkat lunak sebagai sistem pendukung yang membangun editor ini adalah sebagai berikut :

- Sistem operasi Windows 95 versi 4.00.950 B
- Bahasa pemrograman DELPHI 3
- Application Programming Interface OpenGL

Berikut ini dijelaskan hasil perhitungan kinematika invers untuk penentuan joint-joint yang mempunyai posisi awal seperti pada gambar 5.1. Langkah-langkah yang dilakukan berupa perhitungan terhadap tenaga tarikan, penentuan isi matrik Jacobian, perhitungan theta sebagai hasil perkalian matrik Jacobian dengan tenaga

tarikan, dan hasil integrasi theta. Visualisasi hasil perhitungan dapat dilihat pada gambar 5.2.



Gambar 5.1 Posisi awal joint sebelum iterasi

Perhitungan hasil ujicoba pertama adalah :

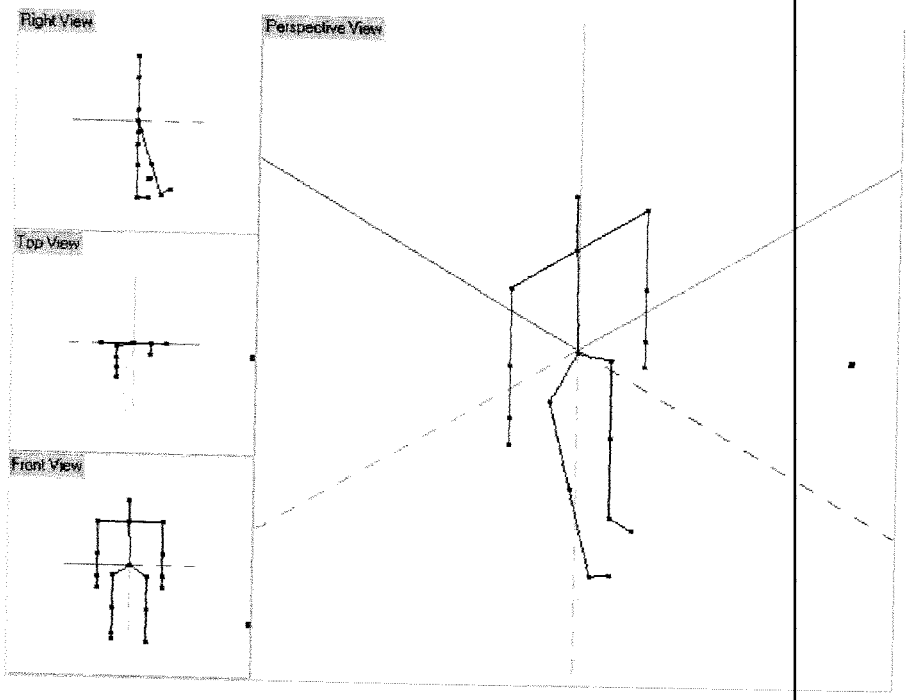
```
Based  index= 14  Nama = Pelvis
EndEff index= 18  Nama = Left_leg
Posisi Yang Dituju x = -5.22580623626709 y = -2.51612901687622 z = 0
Posisi EndEff      x = -6.22853          y = -2.95363          z = 1.5

Iterasi ke = 1
tenaga tarikan
x = 1.12662696838379 y = 0.170042991638184 z = 0
Matrik Jacobian
 -3.26736 6.627009 -1.96961 -1.93442 -0.50792
 -3.64831 8.077829 -4.92403 -4.83606 -1.26982
 -2.50546 3.725368 -7.87846 -7.73770 -2.03171
 -1.36262 -0.62709 -6.89365 -5.93437 -3.59459
 0 0 0 -6.35243 -2.68617
 0 0 0 0 0
Hasil Perkalian Matrik Jacobian dan tenaga tarikan
Theta[14] = -2.55422
Theta[15] = -2.73670
Theta[16] = -2.18925
```

```
Theta[17] = -1.64180
Theta[18] = 0
Setelah diintegral dan ditambahkan dengan theta sebelumnya
Theta[14] = 1.277114
Theta[15] = 1.368353
Theta[16] = 0
Theta[17] = 0
Theta[18] = 0
Sudut theta
Theta[14] = 195.989364624023
 Position[14].x = 0 Position[14].y = 0 Position[14].z = 0
Theta[15] = 1.36835372447968
 Position[15].x = -0.96131 Position[15].y = -0.27545 Position[15].z = 1.5
Theta[16] = 0
 Position[16].x = -3.82469 Position[16].y = -1.17046 Position[16].z = 1.5
Theta[17] = 100
 Position[17].x = -6.68807 Position[17].y = -2.06547 Position[17].z = 1.5
Theta[18] = 90
 Position[18].x = -6.22853 Position[18].y = -2.95363 Position[18].z = 1.5

Iterasi ke = 2
tenaga tarikan
x = 1.00272655487061 y = 0.437504053115845 z = 0
Matrik Jacobian
 -3.60407 6.458860 -2.24732 -1.92262 -0.55091
 -4.01726 7.900830 -5.06289 -4.77915 -1.46880
 -2.67474 3.605756 -7.94567 -7.64253 -2.36381
 -1.33223 -0.68931 -6.88922 -5.76898 -3.84178
 0 0 0 -6.22853 -2.95363
 0 0 0 0 0
Hasil Perkalian Matrik Jacobian dan tenaga tarikan
Theta[14] = -0.78812
Theta[15] = -0.57156
Theta[16] = -1.10450
Theta[17] = -1.63744
Theta[18] = 0
Setelah diintegral dan ditambahkan dengan theta sebelumnya
Theta[14] = 1.671173
Theta[15] = 1.654138
Theta[16] = 0
Theta[17] = 0
Theta[18] = 0
Sudut theta
Theta[14] = 196.383422851562
 Position[14].x = 0 Position[14].y = 0 Position[14].z = 0
Theta[15] = 1.6541383266449
 Position[15].x = -0.95939 Position[15].y = -0.28206 Position[15].z = 1.5
Theta[16] = 0
 Position[16].x = -3.81195 Position[16].y = -1.21098 Position[16].z = 1.5
Theta[17] = 100
 Position[17].x = -6.66451 Position[17].y = -2.13990 Position[17].z = 1.5
Theta[18] = 90
 Position[18].x = -6.19446 Position[18].y = -3.02254 Position[18].z = 1.5

Iterasi ke = 3
tenaga tarikan
x = 0.968661785125732 y = 0.506417036056519 z = 0
Matrik Jacobian
```

```
-3.68762 6.413515 -2.30516 -1.91879 -0.56412
-4.11072 7.852608 -5.09181 -4.76281 -1.52062
-2.71734 3.573766 -7.95963 -7.61537 -2.44954
-1.32395 -0.70507 -6.88823 -5.72441 -3.90518
0          0         0       -6.19446 -3.02254
0          0         0             0      0
Hasil Perkalian Matrik Jacobian dan tenaga tarikan
Theta[14] = -0.32415
Theta[15] = -0.00520
Theta[16] = -0.82236
Theta[17] = -1.63953
Theta[18] = 0
Setelah diintegral dan ditambahkan dengan theta sebelumnya
Theta[14] = 1.833251
Theta[15] = 1.656741
Theta[16] = 0
Theta[17] = 0
Theta[18] = 0
Sudut theta
Theta[14] = 196.545501708984
 Position[14].x = 0 Position[14].y = 0 Position[14].z = 0
Theta[15] = 1.65674114227295
 Position[15].x = -0.95859 Position[15].y = -0.28477 Position[15].z = 1.5
Theta[16] = 0
 Position[16].x = -3.80847 Position[16].y = -1.22189 Position[16].z = 1.5
Theta[17] = 100
 Position[17].x = -6.65835 Position[17].y = -2.15900 Position[17].z = 1.5
Theta[18] = 90
 Position[18].x = -6.18576 Position[18].y = -3.04029 Position[18].z = 1.5

Iterasi ke = 4
tenaga tarikan
x = 0.959961414337158 y = 0.524165391921997 z = 0
Matrik Jacobian
 -3.70611 6.402869 -2.30569 -1.91718 -0.56955
 -4.13327 7.840760 -5.09207 -4.75843 -1.53426
 -2.72760 3.565941 -7.95976 -7.60831 -2.47138
 -1.32192 -0.70887 -6.88822 -5.71318 -3.92157
 0 0 0 -6.18576 -3.04029
 0 0 0 0 0
Hasil Perkalian Matrik Jacobian dan tenaga tarikan
Theta[14] = -0.20156
Theta[15] = 0.142069
Theta[16] = -0.74924
Theta[17] = -1.64056
Theta[18] = 0
Setelah diintegral dan ditambahkan dengan theta sebelumnya
Theta[14] = 1.934036
Theta[15] = 1.585706
Theta[16] = 0
Theta[17] = 0
Theta[18] = 0
Sudut theta
Theta[14] = 196.646286010742
 Position[14].x = 0 Position[14].y = 0 Position[14].z = 0
Theta[15] = 1.58570635318756
 Position[15].x = -0.95809 Position[15].y = -0.28646 Position[15].z = 1.5
Theta[16] = 0
```

```
 Position[16].x = -3.80748 Position[16].y = -1.22505 Position[16].z = 1.5
Theta[17] = 100
 Position[17].x = -6.65687 Position[17].y = -2.16365 Position[17].z = 1.5
Theta[18] = 90
 Position[18].x = -6.18383 Position[18].y = -3.04469 Position[18].z = 1.5

Iterasi ke = 5
tenaga tarikan
x = 0.958027362823486 y = 0.528564691543579 z = 0
Matrik Jacobian
 -3.70765 6.401475 -2.29131 -1.91618 -0.57292
 -4.13734 7.838613 -5.08488 -4.75728 -1.53792
 -2.72945 3.564524 -7.95629 -7.60667 -2.47651
 -1.32155 -0.70956 -6.88847 -5.71079 -3.92573
 0 0 0 -6.18383 -3.04469
 0 0 0 0 0
Hasil Perkalian Matrik Jacobian dan tenaga tarikan
Theta[14] = -0.16843
Theta[15] = 0.179522
Theta[16] = -0.73080
Theta[17] = -1.64114
Theta[18] = 0
Setelah diintegral dan ditambahkan dengan theta sebelumnya
Theta[14] = 2.018249
Theta[15] = 1.495944
Theta[16] = 0
Theta[17] = 0
Theta[18] = 0
Sudut theta
Theta[14] = 196.730499267578
 Position[14].x = 0 Position[14].y = 0 Position[14].z = 0
Theta[15] = 1.49594497680664
 Position[15].x = -0.95766 Position[15].y = -0.28787 Position[15].z = 1.5
Theta[16] = 0
 Position[16].x = -3.80715 Position[16].y = -1.22619 Position[16].z = 1.5
Theta[17] = 100
 Position[17].x = -6.65663 Position[17].y = -2.16451 Position[17].z = 1.5
Theta[18] = 90
 Position[18].x = -6.18367 Position[18].y = -3.04559 Position[18].z = 1.5

Iterasi ke = 6
tenaga tarikan
x = 0.95787239074707 y = 0.529466390609741 z = 0
Matrik Jacobian
 -3.70478 6.402509 -2.27315 -1.91533 -0.57574
 -4.13658 7.839014 -5.07580 -4.75698 -1.53896
 -2.72910 3.564788 -7.95191 -7.60646 -2.47728
 -1.32162 -0.70943 -6.88878 -5.71072 -3.92668
 0 0 0 -6.18367 -3.04559
 0 0 0 0 0
Hasil Perkalian Matrik Jacobian dan tenaga tarikan
Theta[14] = -0.15879
Theta[15] = 0.188171
Theta[16] = -0.72670
Theta[17] = -1.64157
Theta[18] = 0
Setelah diintegral dan ditambahkan dengan theta sebelumnya
Theta[14] = 2.097640
```

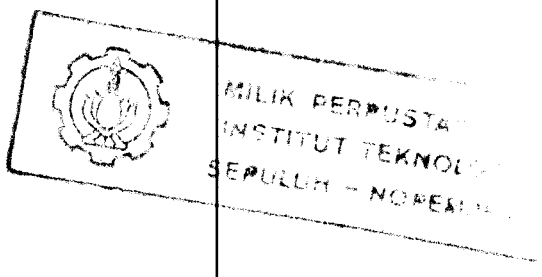
```
Theta[15] = 1.401859
Theta[16] = 0
Theta[17] = 0
Theta[18] = 0
Sudut theta
Theta[14] = 196.80989074707
 Position[14].x = 0 Position[14].y = 0 Position[14].z = 0
Theta[15] = 1.40185928344727
 Position[15].x = -0.95726 Position[15].y = -0.28919 Position[15].z = 1.5
Theta[16] = 0
 Position[16].x = -3.80699 Position[16].y = -1.22678 Position[16].z = 1.5
Theta[17] = 100
 Position[17].x = -6.65671 Position[17].y = -2.16437 Position[17].z = 1.5
Theta[18] = 90
 Position[18].x = -6.18398 Position[18].y = -3.04558 Position[18].z = 1.5
```



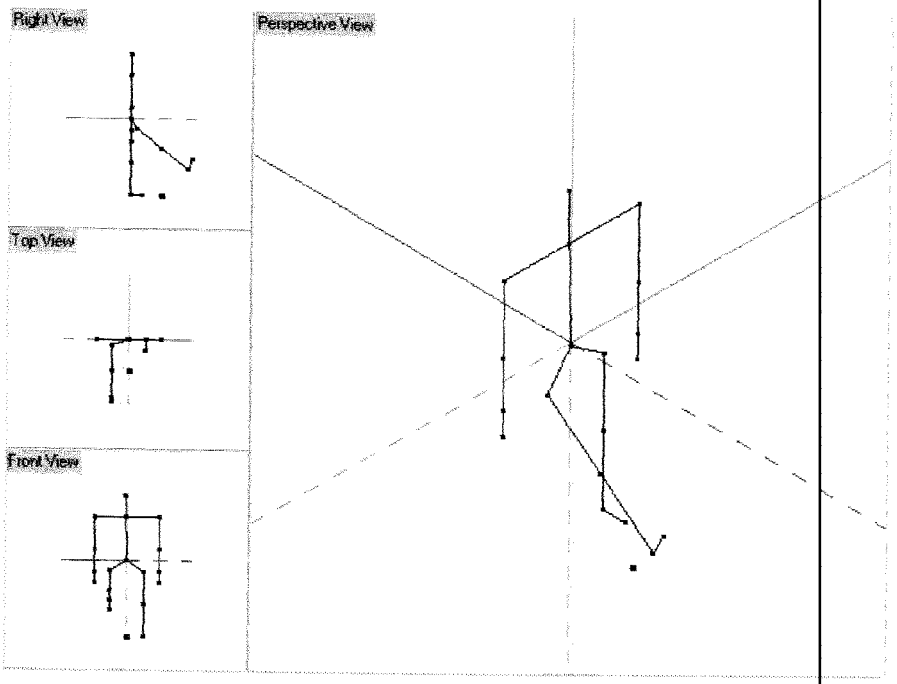
Gambar 5.2 Posisi hasil percobaan pertama setelah iterasi

Perhitungan ujicoba yang kedua dilakukan dengan menggunakan posisi awal seperti pada gambar 5.1 dengan detail yang sama seperti percobaan pertama membutuhkan 14 iterasi. Sedang hasil visualisasinya dapat dilihat pada gambar 5.3.

```
Based  index = 14  Nama =Pelvis
EndEff index = 18  Nama =Left_leg
Posisi Yang Dituju  x = -3.03225803375244 y = -4.5806450843811 z = 0
Posisi EndEff       x = -3.68150          y = -5.69512         z = 1.5

Iterasi ke = 1
tenaga tarikan
x = 0.674095869064331 y = 1.10093307495117 z = 0
Matrik Jacobian
 -7.02236 2.961454 -6.13442 -1.73205 -1
 -7.77236 4.260492 -7.00644 -3.34643 -3.63808
 -4.24202 1.470034 -8.85277 -5.20673 -5.99164
 -0.71167 -1.32042 -6.75987 -2.82607 -6.15602
 0 0 0 -3.70635 -5.68157
 0 0 0 0 0
Hasil Perkalian Matrik Jacobian dan tenaga tarikan
Theta[14] = -1.47338
Theta[15] = -0.54880
Theta[16] = -1.24111
Theta[17] = -1.93343
Theta[18] = 0
Setelah diintegral dan ditambahkan dengan theta sebelumnya
Theta[14] = 0
Theta[15] = 0.274400
Theta[16] = 0
Theta[17] = 0
Theta[18] = 0
Sudut theta
Theta[14] = 210
 Position[14].x = 0 Position[14].y = 0 Position[14].z = 0
Theta[15] = 21.9508323669434
 Position[15].x = -0.86602 Position[15].y = -0.5 Position[15].z = 1.5
Theta[16] = 0
 Position[16].x = -2.71503 Position[16].y = -2.86244 Position[16].z = 1.5
Theta[17] = 100
 Position[17].x = -4.56405 Position[17].y = -5.22489 Position[17].z = 1.5
Theta[18] = 90
 Position[18].x = -3.68150 Position[18].y = -5.69512 Position[18].z = 1.5
.
.
.

Iterasi ke = 14
tenaga tarikan
x = 0.604620456695557 y = 1.13841533660889 z = 0
Matrik Jacobian
 -7.07859 2.857241 -6.26882 -1.73205 -1
 -7.82859 4.156279 -7.07364 -3.30428 -3.67094
 -4.26128 1.413233 -8.88282 -5.13298 -6.04914
 -0.69397 -1.32981 -6.75278 -2.75033 -6.18170
 0 0 0 -3.63687 -5.71906
 0 0 0 0 0
Hasil Perkalian Matrik Jacobian dan tenaga tarikan
Theta[14] = -1.02713
Theta[15] = -0.00175
Theta[16] = -0.96761
Theta[17] = -1.93346
Theta[18] = 0
```



```
Setelah diintegral dan ditambahkan dengan theta sebelumnya
Theta[14] = 0
Theta[15] = 0.766332
Theta[16] = 0
Theta[17] = 0
Theta[18] = 0
Sudut theta
Theta[14] = 210
 Position[14].x = 0 Position[14].y = 0 Position[14].z = 0
Theta[15] = 22.4427642822266
 Position[15].x = -0.86602 Position[15].y = -0.5 Position[15].z = 1.5
Theta[16] = 0
 Position[16].x = -2.69468 Position[16].y = -2.87823 Position[16].z = 1.5
Theta[17] = 100
 Position[17].x = -4.52334 Position[17].y = -5.25646 Position[17].z = 1.5
Theta[18] = 90
 Position[18].x = -3.63679 Position[18].y = -5.71910 Position[18].z = 1.5
```

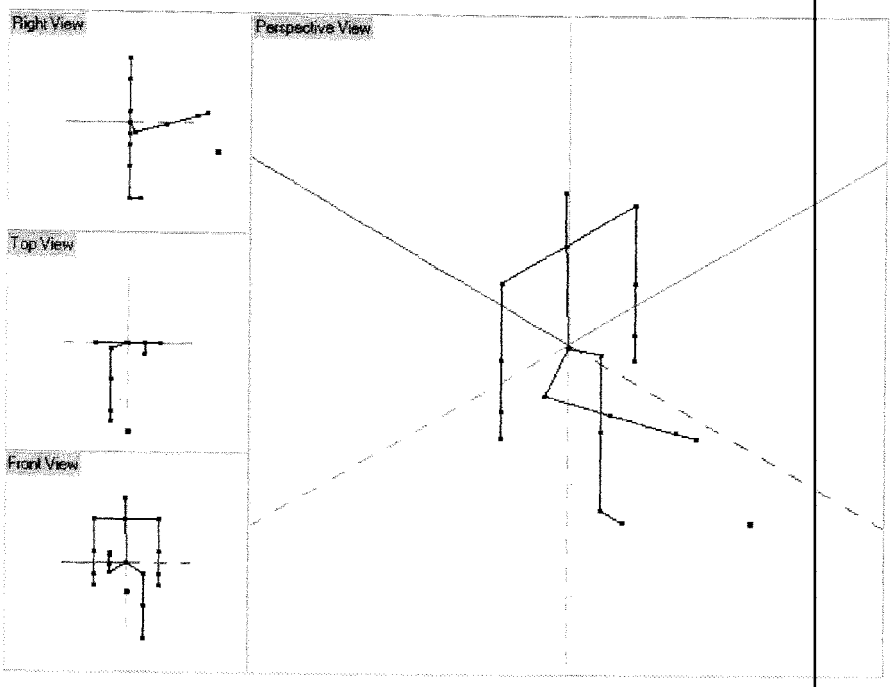


Gambar 5.3 Posisi hasil percobaan kedua setelah iterasi

Perhitungan hasil ujicoba yang ketiga dengan menggunakan posisi awal seperti pada gambar 5.1 adalah menghasilkan iterasi sebanyak 168 dengan Posisi akhir seperti pada gambar 5.4

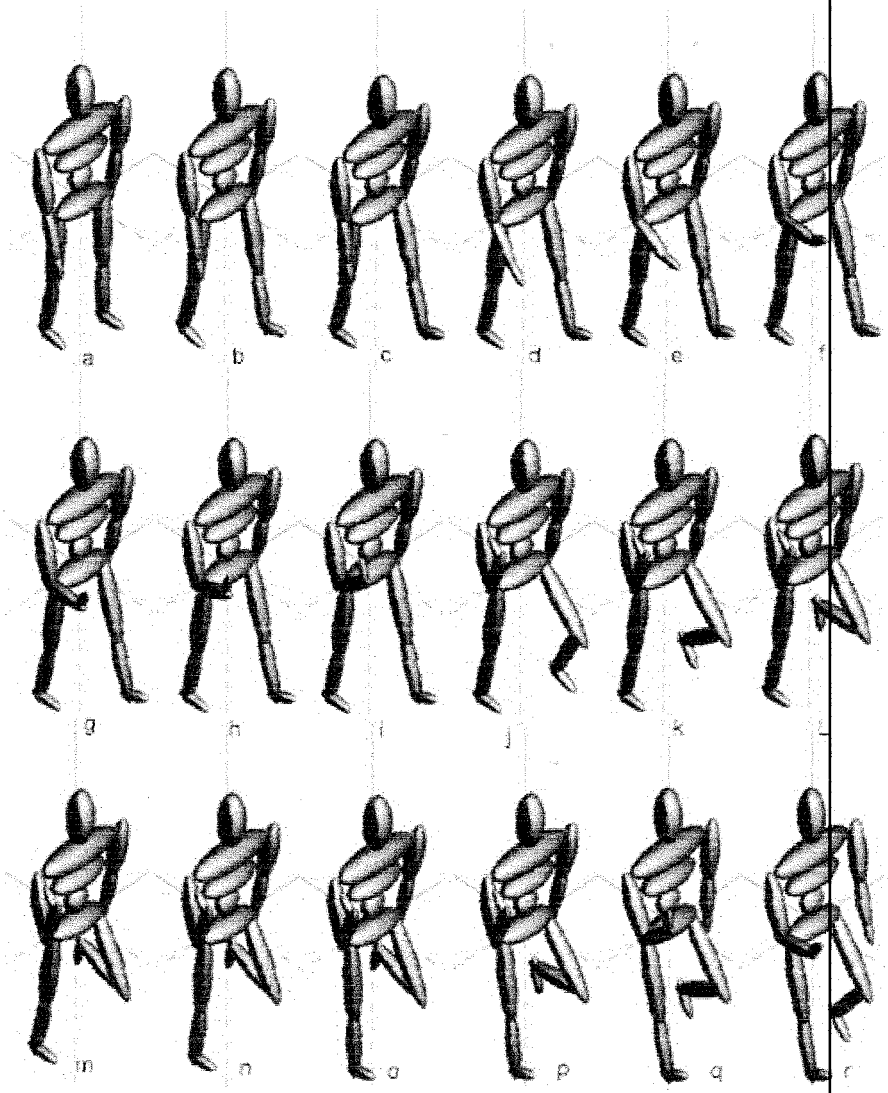
Dari hasil tersebut dapat dilihat bahwa semakin jauh posisi *end-effector* yang

dimasukkan maka jumlah iterasi akan semakin banyak. Dan karena penentuan posisi *end-effector* dilakukan oleh pengguna dengan cara menggerakkan mouse maka ini berarti semakin cepat pengguna menggerakkan mouse maka posisi *end-effector* baru yang dihitung akan semakin jauh dan ini mengakibatkan iterasi yang terjadi semakin banyak dan proses semakin lambat.

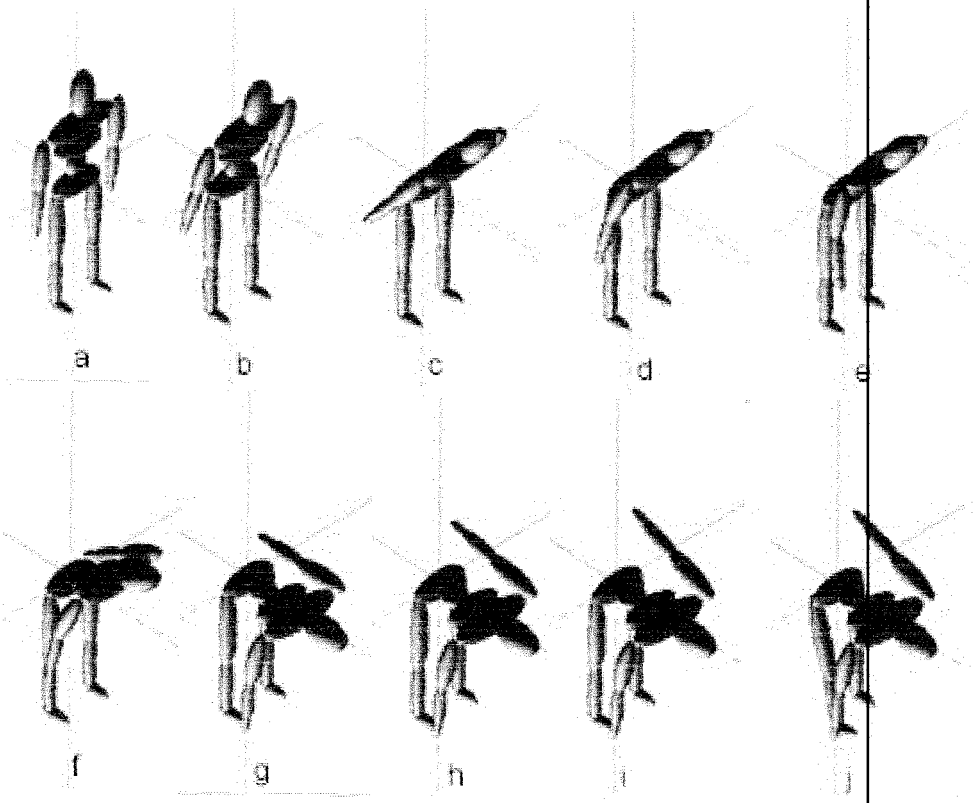


Gambar 5.4 Posisi hasil percobaan ketiga setelah iterasi

Dari empat ruang pandang yang disediakan untuk memasukkan posisi end-effector dari pengguna dapat dihasilkan pose-pose atau posisi manusia. Dari sejumlah pose posisi tersebut dapat dilakukan animasi akhir yang memperlihatkan pergerakan manusia.



Gambar 5.5 Contoh sequence frame untuk animasi



Gambar 5.6 Contoh lain sequence frame untuk animasi

Gambar 5.5 dan 5.6 menunjukkan berbagai macam pose yang dihasilkan pengguna dari masing-masing antarmuka seperti yang dijelaskan dalam Lampiran.

BAB VI

PENUTUP

Dalam bab ini diuraikan hal-hal yang menjadi kesimpulan dari hasil uji coba perangkat lunak, serta beberapa saran yang diharapkan di waktu mendatang dapat menyempurnakan perangkat lunak ini sehingga semakin berguna di bidang ilmu grafika komputer.

6.1 Kesimpulan

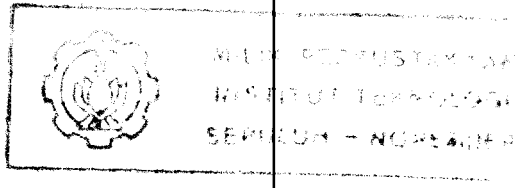
Beberapa kesimpulan yang dapat ditarik dari hasil uji coba perangkat lunak adalah sebagai berikut :

1. Penyelesaian kinematika invers memiliki perhitungan yang relatif sederhana. Hal ini karena tidak perlu menghitung invers untuk matrik jacobian pada setiap iterasi. Kecepatan perhitungan kinematika invers tergantung pada besarnya jarak antara posisi end-effector lama dan posisi end-effektor yang baru. Semakin jauh jaraknya maka respon yang dihasilkan akan semakin lama , karena iterasi yang dilakukan semakin banyak.
2. Penggunaan fungsi-fungsi OpenGL sangat membantu dalam visualisasi hasil. Dimana tidak perlu dilakukan perhitungan konversi koordinat dari tiga dimensi menjadi koordinat layar maupun perhitungan untuk pewarnaan benda karena semuanya telah ada dalam perintah OpenGL. Sehingga konsentrasi dapat lebih ditekankan pada perhitungan iterasi kinematika invers.

6.2 Saran Pengembangan Lebih Lanjut

Beberapa kemungkinan pengembangan lebih lanjut yang dapat dilakukan pada perangkat lunak ini adalah sebagai berikut :

1. Jenis obyek yang dianimasikan dapat dibuat lebih fleksibel dan tidak terbatas pada manusia saja. Obyek dapat dimasukkan dan dapat diubah parameternya secara langsung.
2. Dimasukkan perhitungan-perhitungan yang mampu membuat animasi lebih realistis, seperti unsur gravitasi, momentum, dan kecepatan
3. Karena adanya batasan perputaran joint pada sumbu x saja maka hasil akhir memiliki jenis pergerakan dengan arah memutar pada sumbu x. Sebagai pengembangannya dapat tambahan dibuat perputaran pada sumbu y dan sumbu z. Sehingga kemungkinan posisi pergerakannya dapat dibuat lebih banyak dan realistik.
4. Dengan digunakan fungsi-fungsi *library* dari OpenGL maka pembentukan obyek dapat dibuat lebuh kompleks dengan menggunakan fasilitas-fasilitas yang telah disediakan sehingga obyek yang dianimasikan bukan dalam bentuk kerangka saja tetapi dapat dikembangkan menjadi kerangka yang memiliki daging, kulit bahkan baju.



DAFTAR PUSTAKA

1. Chin, Kwan. *"Closed Form And Generalized Inverse Kinematic Solutions for Animating Human Articulated Structure"*, Curtin University Of Technology, 1996
2. Fu , K.S , Gonzales, R.C. , Lee ,C.G, *"Robotics"*, McGraw-Hill ,New York,1988
3. Hill , Francis S, "*Computer Graphics*", Maxwell Macmillan,New York, 1996
4. Neider Jackie, Davis Tom , Mason Woo, "*OpenGL Programming Guide*"
5. Piegl Les, Tiller W, "*The NURBS Book (Second Edition)*", Springer , 1995
6. Watt, Allan, "*Advanced Animation And Rendering Techniques*", Adison Wesley Publisher Ltd, New York, 1992
7. Welman, Chris , *"Inverse kinematics and Geometric Constraint For Articulated Figure Manipulation"*, Simon Frasier University, 1993
8. Wright Richard S. , Sweet Michael ,"*OpenGL Super Bible*", Waite Group Press, 1996

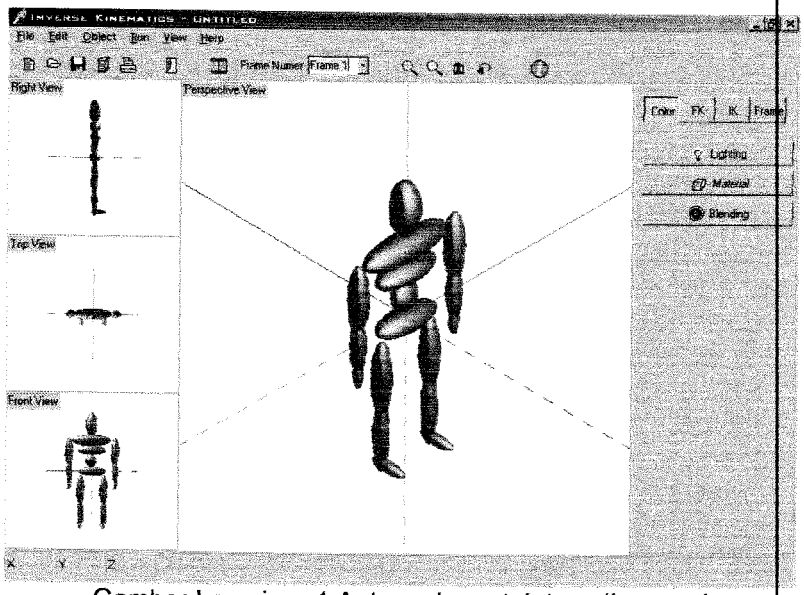
LAMPIRAN

PETUNJUK PENGUNAAN PERANGKAT LUNAK

Perangkat lunak ini memiliki dua macam menu utama pada tampilan awal ruang pandang utama yaitu menu atas (*toolbar*) dan menu samping. Menu atas terdiri dari pilihan File, Edit, Obiect, View, dan Help. Sementara itu menu samping terdiri dari pilihan Color, FK , IK , dan Frame seperti yang ditunjukkan Gambar lampiran 1

Menu File berhubungan dengan berkas teks masukan, dan terdiri dari lima buah submenu yaitu :

- New, untuk membuat file animasi baru.
- Open, untuk membuka berkas file teks yang ada. Pemakai dapat memasukkan nama dan lokasi berkas melalui jendela dialog yang ditampilkan.
- Save, untuk menyimpan data animasi yang sedang aktif ke dalam berkas file teks. Penyimpanan dilakukan melalui sebuah jendela dialog dengan memasukkan nama file dan lokasinya.
- Save frame As, untuk menyimpan data animasi pada frame yang sedang aktif ke dalam file bitmap atau jpeg. Seperti pada sub menu Save penyimpanan juga dilakukan melalui sebuah jendela dialog dengan memasukkan nama file dan lokasinya.
- Print frame, untuk mencetak gambar dari frame yang sedang aktif pada layar melalui media printer.
- Exit, untuk berhenti dari program.



Gambar Lampiran 1 Antarmuka untuk tampilan awal

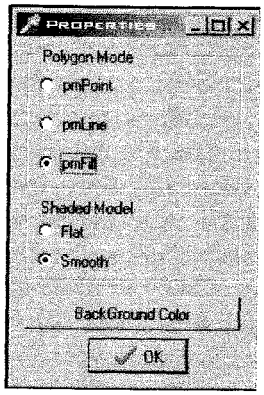
Di dalam menu Edit terdapat sub menu undo yang memberikan kesempatan kepada pengguna untuk kembali pada kondisi sebelum perintah diberikan. Batas kinerja undo adalah lima kali.

Menu object adalah menu yang memberikan pilihan untuk tampilan manusia yang ada. Apabila tidak dilakukan pemilihan maka itu berarti pemakai hanya mendapatkan tampilan kerangka manusia. Ada dua macam tampilan manusia yang disediakan yaitu bentuk bundaran dan cilinder. Untuk bentuk bundaran dapat dilihat pada gambar Lampiran 1 dan bentuk cilinder dapat dilihat pada Lampiran 3. Pilihan properties berguna untuk mengatur tampilan layar benda. Isinya berupa pilihan tampilan manusia dan background dari layar seperti pada gambar Lampiran 2.

Menu Run berisi perintah untuk memulai menjalankan animasi manusia

Menu View memberikan pilihan tampilan jendela yang berbeda. Terdapat dua jenis tampilan , yaitu normal dan classic. Untuk tampilan normal pemakai akan

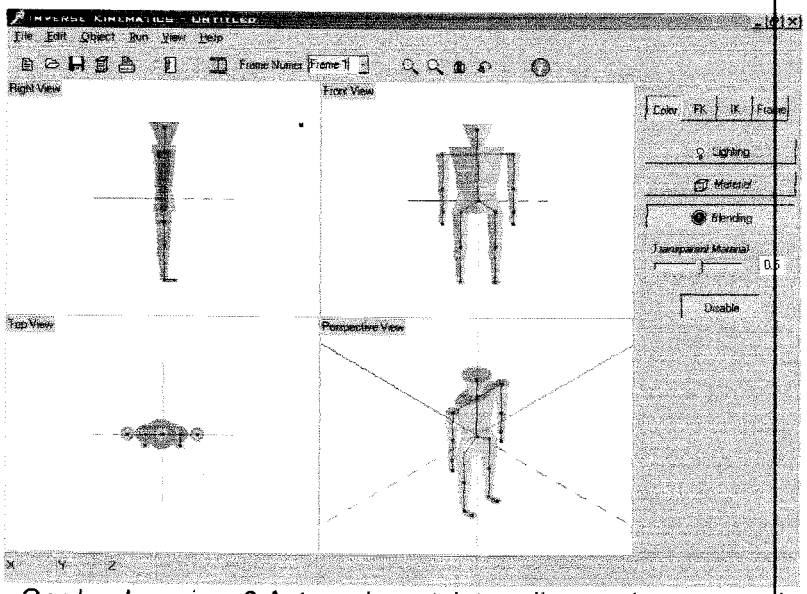
mendapatkan empat macam ruang pandang yaitu tampak depan (*front view*), tampak samping kanan (*right view*), tampak atas (*top view*) dan perspektif (*perspective view*) dengan ukurannya masing-masing sama. Hal ini dapat dilihat pada gambar lampiran 3.



Lampiran 2 Tampilan menu untuk properties

Sedangkan dalam tampilan *classic* pemakai juga mendapat empat ruang pandang yaitu satu ruang pandang perspektif yang besar dan tiga lainnya yang berukuran kecil seperti ditunjukkan pada gambar lampiran 1. Selain itu di dalam Menu View terdapat sub menu persepktif/ orto untuk memberikan pilihan tampilan perspektif atau ortogonal. Dan dua sub menu terakhir dalam menu view adalah zoom in dan zoom out yang berguna untuk memperbesar dan memperkecil tampilan gambar.

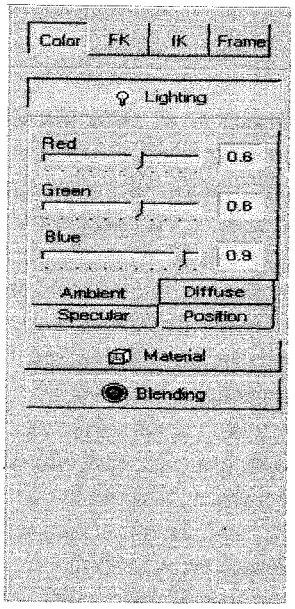
Menu samping merupakan menu khusus yang berhubungan dengan manipulasi obyek manusia. Menu yang terdiri dari empat sub menu ini dapat diakses dengan menekan *button* yang sesesuaian.



Gambar Lampiran 3 Antarmuka untuk tampilan awal cara normal

Button color akan menampilkan suatu menu yang berisi *button* lighting , material dan blending. Penekanan terhadap *button* lighting akan menampilkan sub menu lighting. Di dalam sub menu lighting terdapat tab-tab yang mewaliki pencahayaan pada OpenGL, yaitu nilai *Ambient, Diffuse, Specular,* dan posisi cahaya. Untuk *Ambient, Diffuse,* dan *Specular* akan memiliki masukan warna *red, green, blue* yang diwakili dengan *track bar* sedangkan untuk posisi cahaya, masukannya berupa kotak edit untuk posisi X, Y, dan Z. Menu lighting ini dapat dilihat pada gambar lampiran 4. Sedangkan sub menu material pada gambar lampiran 5 akan muncul bila dilakukan penekanan terhadap *button* Material. Sama seperti sub menu lighting, di dalam sub menu material ini terdapat tab-tab yang mewaliki pemberian properti material yang dilakukan OpenGL. Perbedaannya adalah di dalam material tidak terdapat tab *position* melainkan diganti dengan tambahan properti *shininess*, yang merupakan nilai kilau pada obyek manusia.

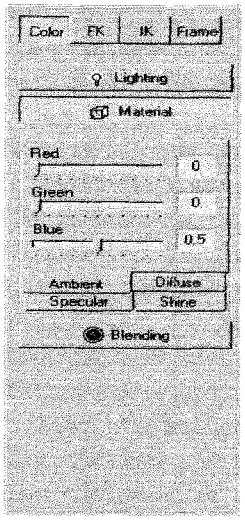
Button yang terakhir adalah bending, yang menampilkan menu untuk mengatur nilai transparansi dari obyek manusia seperti pada gambar lampiran 6. Perlu diingat bahwa obyek manusia dapat ditampilkan dengan salah satu cara yaitu solid atau transparan. Apabila obyek tampak solid maka ini berarti blending tidak berfungsi dan sebaliknya.



Gambar lampiran 4 Tampilan Menu *Color* untuk Pencahayaan

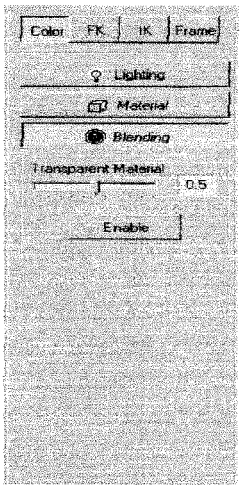
Button FK (*forward kinematics*) mewaliki *toolbox* yang diperlukan untuk proses perhitungan kinematika maju seperti terlihat pada gambar lampiran 7 . Di dalamnya terdapat sebuah *button* bertuliskan *choose joint to move*, yang berfungsi untuk menunjukkan hasil status pengeclickan pada layar. Artinya : untuk memilih *joint* mana yang hendak digerakkan dapat dilakukan dengan menekan *button* ini diikuti dengan mengeclick posisi joint pada salah satu ruang pandang (depan , samping kanan, atau atas). Selama button ini dalam posisi *down* maka setiap pengelickan pada layar akan diperiksa apakah ada joint yang terpilih. Setelah

dilakukan pemilihan maka joint tersebut dapat digerakkan dengan metode kinematika maju dengan mengubah parameter theta. Dalam *toolbox* FK juga ditampilkan range theta yang diperbolehkan dan juga posisi joint hasil perhitungan kinematika maju.

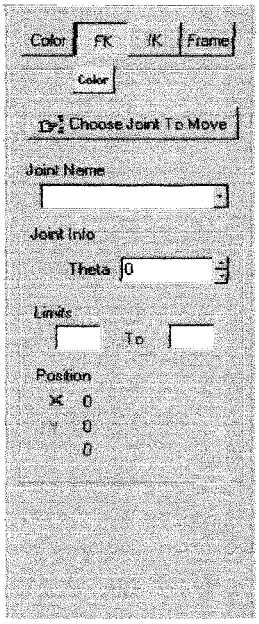


Gambar lampiran 5 Tampilan Menu *Color* untuk Material

Button IK (*Inverse kinematics*) mewaliki *toolbox* yang diperlukan untuk proses perhitungan kinematika invers seperti terlihat pada gambar lampiran 8. Di dalamnya juga terdapat sebuah *button* bertuliskan *choose joint to move*, yang berfungsi untuk menunjukkan bahwa proses pemilihan joint dapat dilakukan. Untuk setiap joint yang dipilih akan memiliki sejumlah *parent*. *Parent* adalah joint-joint yang posisinya berada diantara *root* dan joint yang dipilih. Pada toolbox ini ditampilkan hasil perhitungan kinematika invers pada joint yang dipilih berupa posisi akhirnya.



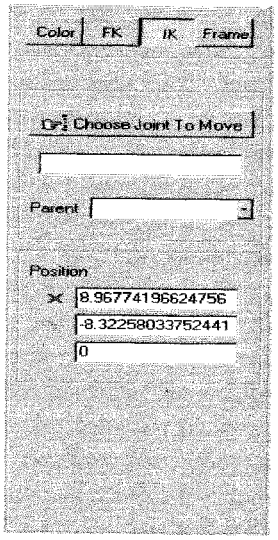
Gambar lampiran 6 Tampilan Menu *Color* untuk Blending



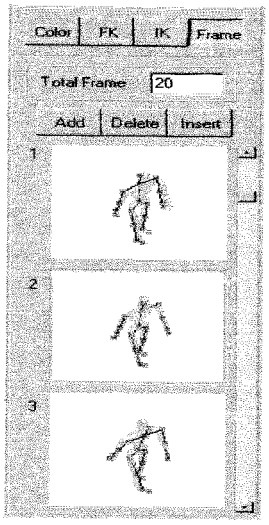
Gambar lampiran 7 Tampilan Menu FK

Penekanan terhadap *button* Frame menampilkan *toolbox* yang diperlukan untuk manipulasi frame-frame yang akan dianimasikan. Di dalamnya terdapat informasi mengenai jumlah frame yang sudah ada. Selain itu terdapat sejumlah *button* tambahan untuk melakukan operasi-operasi pada frame seperti Add, Delete,

dan Insert. *Button* Add untuk menambahkan frame pada posisi akhir deretan frame-frame yang sudah ada. *Button* Delete untuk menghabus frame yang sedang aktif dan *button* Insert untuk menambah frame pada tengah-tengah urutan yang sudah ada. Selain itu di dalam *toolbox* ini juga terdapat tiga buah *preview* yang memberikan gambar mengenai sebuah frame.

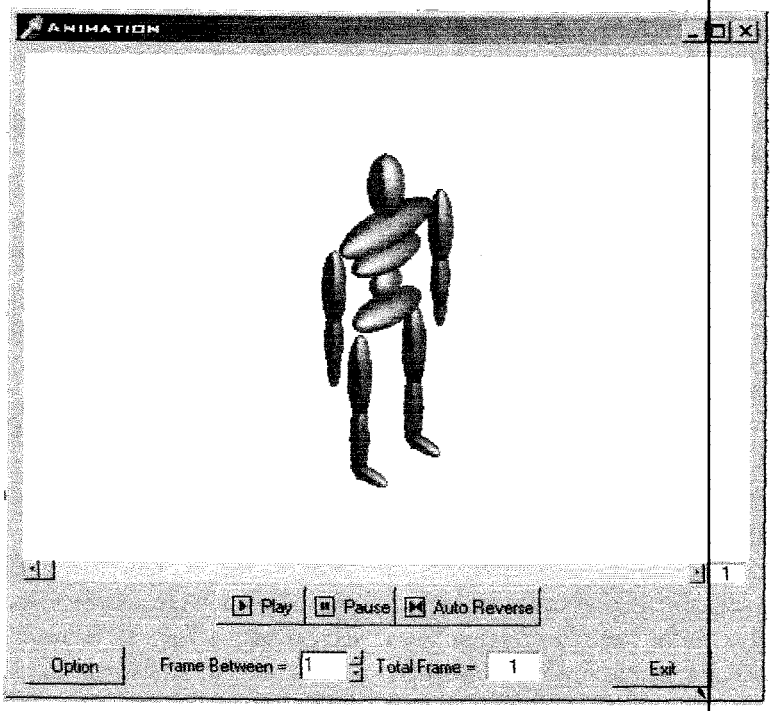


Gambar lampiran 8 Tampilan Menu IK

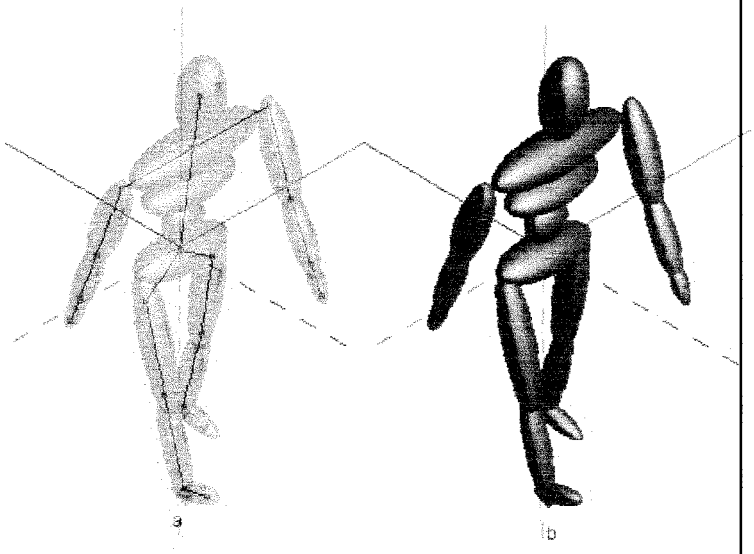


Gambar lampiran 9. Tampilan Menu Frame

Sedangkan Untuk menampilkan hasil akhir frame-frame ini dianimasikan pada suatu form tersendiri yaitu form animasi . Pada form animasi ini terdapat lima buah *button* seperti terlihat pada gambar lampiran 10. *Button* Play adalah untuk memutar animasi, Pause untuk menghentikan animasi pada posisi tertentu , autoreverse untuk menentukan apakah pemutarannya dilakukan berulang, option untuk mengatur tampilan hasil dan Exit untuk keluar dari form animasi dan kembali ke form utama. Dalam menu option terdapat pilihan *transparent* untuk memilih tampilan jenis keluaran obyek manusia. Perbedaan antara obyek manusia solid dan transparan dapat dilihat pada gambar lampiran 11, yaitu pada gambar lampiran 11a menunjukkan obyek manusia transparan dan pada gambar lampiran 11b menunjukkan obyek manusia solid.



Gambar lampiran 10 Tampilan Form Animasi



Gambar lampiran 11 Perbedaan Tampilan Obyek Manusia Solid dan Transparan