BAB III

ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

Analisis merupakan penguraian dari suatu sistem yang utuh ke dalam bagian-bagian komponen dengan maksud untuk mengidentifikasi dan mengevaluasi permasalahan, sehingga diperoleh solusi. Analisis merupakan tahapan yang paling penting, karena kesalahan dalam tahap ini akan menyebabkan kesalahan di tahap selanjutnya. Sistem yang dibuat merupakan program untuk medeteksi sebuah *marker* sekaligus menapilkan objek tiga dimensi yang telah dibuat dengan menggunakan *software* tiga dimensi (3D Max). Objek yang dibuat merupakan sebuah simulasi gerhana, seolah-olah pengguna berinteraksi langsung dengan objek virtual dalam dunia nyata yang disajikan dalam bentuk buku.

3.1.1 Analisis Kebutuhan Non Fungsional

Sistem yang dibangun ini memiliki spesifikasi sebagai berikut:

1) PC atau laptop dengan *processor* 2 GHz atau lebih
2) *Random Access Memory* (RAM) 1 GB
3) *Harddisk* dengan space 20 GB
4) VGA 256 MB 32 Byte
5) *Webcam* 30 fps 5 Mega Pixel
6) Printer Epson C90
7) *Marker*

3.1.2 Analisis Kebutuhan Fungsional

Perangkat lunak yang digunakan untuk simulasi ARTatasurya ini adalah sebagai berikut:

1) *Microsoft window* XP SP2
2) *Autodesk 3D Studio Max 9*
3) ArToolKit *Software Library* Versi 2.65
4) *Adobe Photoshop* CS3
5) OpenGL
6) Visual C++ *Enterprise* 2008

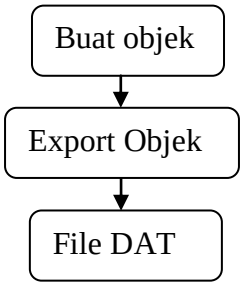
3.1.3 Analisis Sistem yang Sedang Berjalan

Dalam sebuah sistem dibutuhkan analisis masukan dan keluaran agar terjadi interaksi antara pengguna dan sistem. Langkah awal yang harus dilakukan adalah mendapatkan *input* video dari sebuah kamera. Video yang di *streaming* secara *real time* ini akan diolah oleh sistem untuk dianalisa *frame* per *frame.* Analisis ini merupakan proses-proses yang terjadi dalam membuat simulasi tata surya.

3.2.1.1 Proses Pembuatan Objek

Objek yang dibuat menggunakan *software* 3D MAX 9. Objek-objek yang dibuat dalam ARTatasurya ini masih menggunakan *standard primitive.* Dalam proses pemodelan terdiri dari 4 langkah. Pertama, kita membuat objek 3D dengan menggunakan 3D MAX 9. Kedua mengeksport objek yang sudah kita buat ke

dalam ArToolkit. Ketiga, menambahkan file DAT ke dalam ArToolkit. Untuk lebih jelasnya dapat dilihat pada gambar 3.1 dibawah ini.



Gambar 3.1 Tahapan pemodelan

3.2.1.2 Proses Edit data

a. Edit file wrl

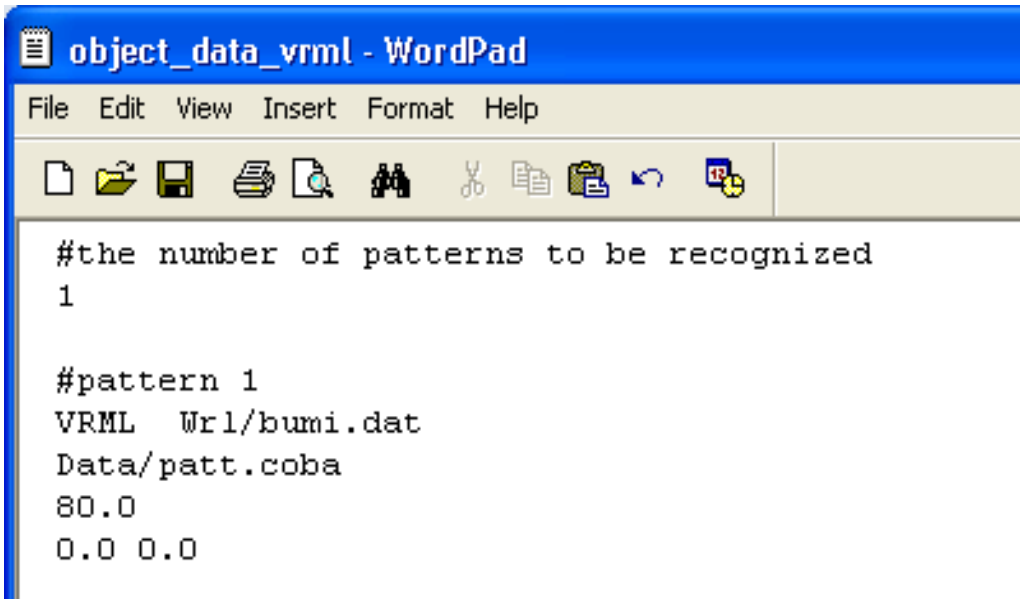
Proses edit data yaitu menambahkan *file* dengan extensi .DAT dengan menggunakan *text editor* wordpad kedalam library ARTatasurya. File dat ini berfungsi untuk memanggil *file* wrl. Wrl ini sendiri merupakan objek yang dibuat dengan menggunakan 3D max. Kode yang ada di dalam bumi.dat.

```
bumi.WRL          \\ memanggil file bumi.wrl
0.0 0.0 0.0             # Translation \\ untukmenggeser objek
90.0 1.0 0.0 0.0        # Rotation  \\ untuk memutar objek
10.0 10.0 10.0          # Scale \\ untuk merubah jarak objek
```

File bumi.dat harus berada dalam satu *folder* dengan *file* bumi.wrl karena dua *file* ini akan saling berhubungan, *file* bumi.dat akan memanggil objek bumi.wrl.

b. Edit objek_data_vrml

File ini berfungsi sebagai tempat objek-objek yang akan di *render* pada *marker* . Cara edit file ini adalah dengan membukanya dengan *WordPad* atau bisa juga dengan menggunakan *Notepad* .



Gambar 3.2 Edit file objek_data_vrml

3.2.1.3 Kalibrasi Kamera

Kalibrasi kamera adalah salah satu langkah yang harus dilakukan dalam proses rekonstruksi 3D, dimana proses ini diperlukan untuk mendapatkan informasi parameter kamera yang digunakan untuk melakukan transformasi dari 3D (*world coordinate*) menuju ke 2D (*camera coordinate*). Parameter kamera teridiri dari dua yaitu parameter intrinsik dan parameter ekstrinsik yang nantinya akan digunakan untuk melakukan perhitungan, sehingga dapat ditentukan letak dari suatu benda dalam ruang 3D. Parameter intrinsik terdiri dari 4 unsur yaitu:

a) Nilai focus kamera, yaitu jarak antara lensa kamera dengan bidang gambar.

b) Titik pusat proyeksi, yaitu lokasi titik tengah gambar dalam *pixel* koordinat

c) Ukuran *pixel* efektif.

d) Koefisien distorsi, yaitu koefisien tingkat kelengkungan lensa meliputi radial dan tangensial distorsi.

Sedangkan parameter ekstrinsik terdiri dari dua buah matriks, yaitu matriks translasi dan matriks rotasi.

$$R = \begin{bmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{bmatrix}$$

$$T = \begin{bmatrix} t1 \\ t1 \\ t3 \end{bmatrix}$$

Matrik rotasi merupakan gabungan dari rotasi terhadap sumbu x,y,z yang masing-masing berotasi pada sudut Θ.

$$R = \begin{bmatrix} cos\,\Theta & -sin\,\Theta & 0 \\ sin\,\Theta & cos\,\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} cos\,\Theta & 0 & sin\,\Theta \\ 0 & 1 & 0 \\ -sin\,\Theta & 0 & cos\,\Theta \end{bmatrix}$$

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\Theta & -sin\,\Theta \\ 0 & sin\,\Theta & cos\,\Theta \end{bmatrix}$$

Berikut sistem koordinat *Marker* :

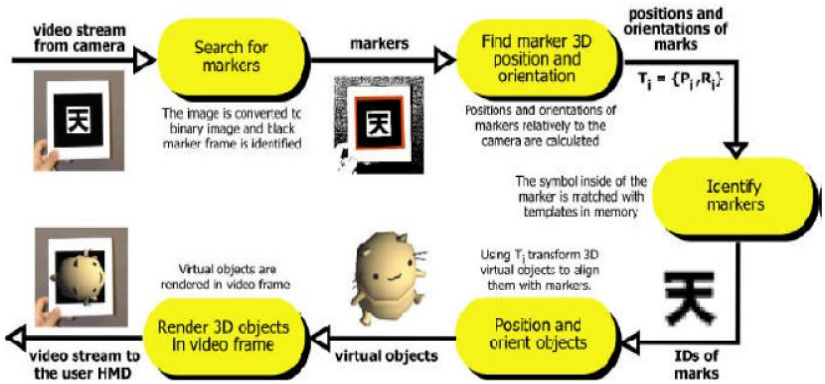
a1x + b1y + c1 = 0 a2x + b2y + c2 = 0 (1)

Parameter pada persamaan diatas akan dipakai pada proses selanjutnya dalam cara kerja ArToolkit.

3.2.1.4 Cara Kerja ArToolkit

ArToolkit menggunakan teknik visi komputer untuk mengkalkulasikan sudut pandang kamera nyata ke *Marker* yang nyata. Ada lima langkah, dalam proses kerja ArTookit.

1. mencari *Marker* , kemudian *Marker* yang dideteksi dirubah menjadi *binary*, kemudian *black frame* atau bingkai hitam terdeteksi oleh kamera.
2. kamera menemukan poisisi *Marker* 3D dan dikalkulasikan dengan kamera nyata.
3. kamera mengindentifikasi *Marker* , apakah pola *Marker* sesuai dengan *templates memory*.
4. Dengan mentrasformasikan posisi *Marker* .
5. Objek 3D di *render* diatas *Marker* .



Gambar 3.3 Cara Kerja ARToolkit

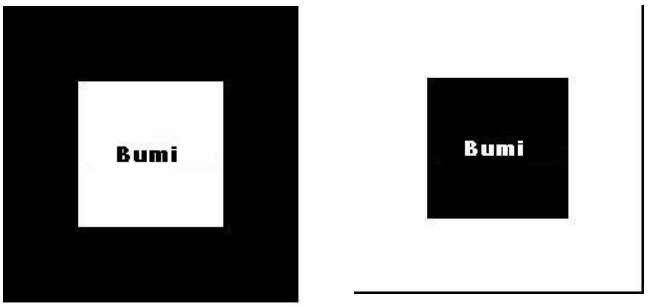
Sebelum kamera digunakan, kamera harus dikalibrasi terlebih dahulu. Kalibrasi kamera merupakan bagian yang sangat penting dalam proses pengambilan *input* video. Hal ini disebabkan oleh distorsi pada lensa kamera yang tiap kamera berbeda karakteristiknya. Tujuan dari kalibrasi kamera adalah untuk menghitung tingkat distorsi dari sebuah lensa kamera yang digunakan agar *image* yang dihasilkan mendekati image ideal. Parameter ini nantinya digunakan dalam perhitungan pada proses *Pose and Position Estimation* agar model objek dapat ditampilkan tepat diatas *marker* .



Image ideal Distorsi *image*

Gambar 3.4 *Image ideal* dan *distorsi image*

Video yang diterima selanjutnya akan mengalami proses binarisasi (*gray-scale),* kemudian nilai *threshold* berada pada angka 0 – 225 dan secara *default*, *thresholding* bernilai 100. Fungsi dari proses ini adalah untuk membantu sistem agar dapat mengenali bentuk segi empat dan pola di *marker* pada video yang diterima. Nilai *threshold* dapat diubah dan disesuaikan dengan kondisi cahaya disekitar *marker* untuk tetap membuat *marker* terlihat sebagai segi empat, karena ketika cahaya disekitar *marker* berkurang ataupun berlebihan pada saat proses *thresholding*, sistem tidak dapat mendeteksi *marker* .



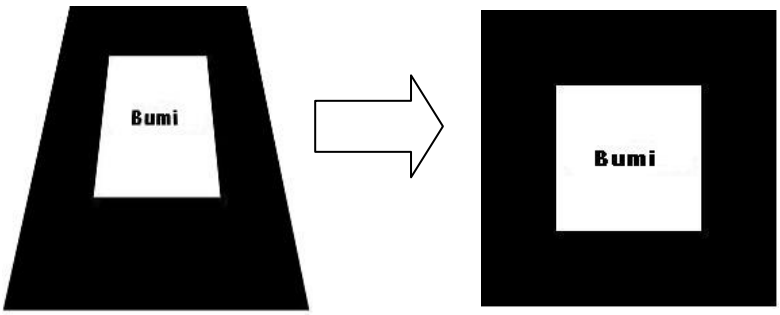
(a) (b)

Gambar 3.5 *Marker* hasil *thresholding*

Setelah video mengalami proses *thresholding* langkah selanjutnya adalah mendeteksi *marker* , dimana sistem akan mengenali bentuk dan pola yang ada pada *marker* . Sistem akan mencari bagian yang memiliki bentuk segi empat dan menandainya. Sistem juga akan menghilangkan area yang tidak berbentuk segi empat sehingga yang akan ditampilkan pada layar hanyalah area yang memiliki bentuk segi empat.

Corner detection digunakan untuk mendapatkan koordinat dari empat sisi dan empat titik sudut pada segi empat yang tersisa. Setelah proses ini selesai, dua garis parallel pada *marker* diproyeksikan sehigga persamaan garisnya pada koordinat layar kamera adalah sperti berikut: $A1x + b1y + c1 = 0$ $a2x + b2y + c2=0$ Parameter pada persamaan diatas akan disimpan dan dipakai pada proses selanjutnya.

Karena sudut dari lensa kamera tidak tegak lurus terhadap *marker* ketika mengambil video, sudut-sudut *marker* yang dibentuk oleh sisi-sisi segi empat 90^o. Ini menyebabkan pola yang ada di dalam *marker* tidak dapat dikenali dengan baik.

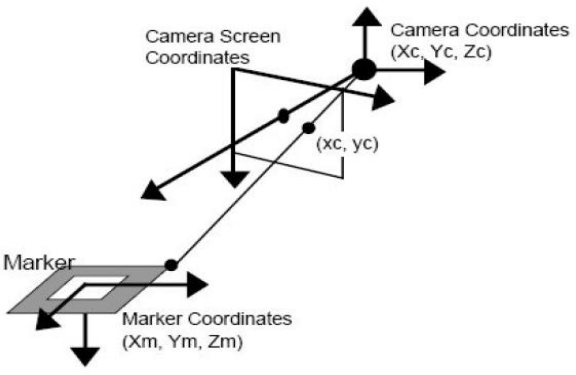


Gambar 3.6 Sudut-sudut pada *marker*

Untuk mengubah sudut *marker* yang tidak 90^o adalah dengan *pattern normalization* agar pola dapat dikenali dan dicocokan. Sebelum gambar dijadikan pola pada sebuah *marker* agar sistem dapat mengenali pola itu. Untuk meletakan objek 3D tepat diatas *marker* , sistem perlu mengetahui koordinat dari *marker* dan kamera.

$$\begin{bmatrix} Xc \\ Yc \\ Zc \\ 1 \end{bmatrix} = \begin{bmatrix} V11 & V12 & V13 & Wx \\ V21 & V22 & V23 & Wy \\ V31 & V32 & V33 & Wz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Xm \\ Ym \\ Zm \\ 1 \end{bmatrix} = \begin{bmatrix} & V3x3 & W3x1 & \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} Xm \\ Ym \\ Zm \\ 1 \end{bmatrix}$$

$$= \text{Tcm} \begin{bmatrix} Xm \\ Ym \\ Zm \\ 1 \end{bmatrix} \quad (2)$$



Gambar 3.7 Sistem koordinat *Marker*

Matriks transformasi (Tcm) dari koordinat *Marker* ke koordinat kamera seperti yang terlihat pada gambar 3.8. untuk *Marker* yang sudah dikenali, nilai dari parameter a1,b1,c1 dan a2,b2,c2 didapatkan ketika proses *contour extraction*. Matrix proyeksi P pada persmaan 3 diperoleh ketika proses kalibrasi kamera. Dengan mengganti xc dan xy pada persamaan 3 untuk x dan y pada persamaan 1 didapat persamaan garis sperti persamaan 4.

$$P = \begin{bmatrix} P11 & P12 & P13 & 0 \\ 0 & P22 & P23 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} hxc \\ hyc \\ h \\ 1 \end{bmatrix} = P \begin{bmatrix} Xc \\ Yc \\ Zc \\ 1 \end{bmatrix} \qquad (3)$$

$$_{a1}P_{11}Xc + (_{a1}P12 + _{b1}P_{22})\ Y_c + _{a1}P_{13} + _{b1}P_{23} + _{c1})\ Z_c=0$$

$$_{a2}P_{11}X_c + (_{a2}P_{12} + _{b12}P_{22})\ Y_c + (_{a2}P_{13} + _{c2})Z_c =0 \qquad (4)$$

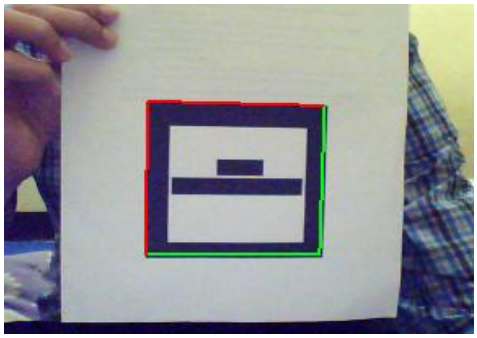
Marker segi empat yang digunakan mempuanyai empat sisi dimana dua sisi adalah garis yang paralel. Vektor normal dari *Marker* adalah n yang dihasilkan dari perkalian *cross vektor* u1 dan u2. Pada kenyataannya, vektor u1 dan u2 seharusnya tegak lurus, hal ini disebabkan oleh sudut kamera ketika pengambilan gambar yang tidak tegak lurus dengan *Marker* . Vektor v1 dan v2 dibuat agar memiliki sudut 90 derajat dengan menggunakan nilai dari vektor u1 dan u2 untuk memperkecil kesalahan. Setelah v1 dan v2 tegak lurus, v3 dihasilkan dari perkalian cross v1 x v2. Nilai v1, v2 dan v3 adalah komponen *Marker* ke koordinat kamera seperti pada persamaan 2.

Seteleah komponen rotasi V3x3 pada matrix transformasi diketahui, komponen translasi W1, W2, W3 dapat diperoleh dengan menggunakan persamaan 2 dan 3. Setelah transformasi matrix didapat, langkah terakhir yang

dilakukan adalah menggambar objek virtual 3D pada frame video tepat diatas permukaan *Marker* dan hasilnya dapat dilihat pada output videonya.

a) Deteksi *Marker*

Deteksi *Marker* merupakan tahap dimana *Marker* akan di identifikasi oleh *webcam* sebagai sasaran untuk penempatan objek yang akan dirender. Deteksi *Marker* dengan menggunakan metode *Hough Transform* mendeteksi parameter-parameter geometri. Representasi garis dari *Marker* yang ditangkap kamera *webcam* menggunakan ($r = x \cos(\theta) + y \sin(\theta)$, r: jarak antar garis dalam kalibrasi kamera, θ: sudut antara garis normal dengan sumbu-x). Input merupakan nilai biner dari titik sudut (*edge*) yang menghubungkan antar garis dimana semua titik sudut tersebut ditentukan sebagai pixel. *Hough Transform* membutuhkan *array* yang disebut *accumulator array* , array ini hanya mempunyai 1 nilai balik untuk setiap kombinasi parameter (r, θ) yang memungkinkan. Setiap garis dapat dibangun dengan menghubungkan antara titik sudut (*edge*) yang telah ditentukan sebagai *pixel* tadi, dan parameter-parameter yang terkait dengan r dan θ menentukan nilai *increment* dari *accumulator array* . Setelah semua garis-garis yang memungkinkan diproses, nilai *array* yang tinggi merepresentasikan sebuah garis (*Marker border*). Meskipun Hough Transform memiliki banyak *noise* dalam sebuah *image* dan belum mampu menemukan titik akhir dari sebuah garis. *Marker* yang berhasil di deteksi akan ditandai dengan warna merah dan warna hijau. Itu menandakan *projection* atau sudut pandang dari *webcam*.



Gambar 3.8 Deteksi *Marker*

3.2.1.5 Langkah Pengembangan ArToolkit

Untuk lebih detailnya ada beberapa langkah dalam pengembangan ArToolkit seprti terlihat pada tabel 3.1.

Tabel 3.1 Prinsip pengembangan *augmented reality*

Inisialisasi	1. Menginisialisasi video capture dan membaca file *marker* dan parameter kamera.
Loop	2. Mengambil video frame input 3. Mendeteksi pola *marker* dan pola gambar dalam kotak hitam pada setiap frame video. 4. Hitung transformasi kamera *relative* terhadap pola yang terdeteksi. 5. Menggambar objek virtual pada pola yang terdeteksi
Penutup	6. Tutup *video capture*

Pada langkah ke dua sampai lima terjadi looping atau pengulangan sampai aplikasi tersebut berhenti, sementara langkah satu dan enam hanya dilakukan pada saat inisialisasi dan pada saat menutup aplikasi. Untuk pengembangan secara rinci

dalam pengembangan ArToolkit dapat dilihat dengan menggunakan Visual Studio 2008 atau versi sebelumnya dengan cara membuka file sumber yang ada dalam direktori ArToolkit/example/SimpleTest/. **S**impleTest merupakan file yang ada dalam ArToolkit yang digunakan sebagai program test yang terdiri dari *main rutin* dan beberapa *rutin graphic* utama. Pada table 3.1, langkah dua sampai dengan lima merupakan fungsi mainloop. Fungsi panggilan dank ode yang sesuai dengan langkah aplikasi ArToolkit dapat dilihat pada table 3.2.

Table 3.2 Fungsi panggilan dalam step ArToolkit

ArToolkit step	Fungsi
1. Menginisialisasi aplikasi	Init
2. Ambil sebuah frame video input	arVideoGetImage (mainloop)
3. Mendeteksi penanda (*marker*)	arDetectMarker (mainloop)
4. Perhitungan kamera transformasi	arGetTransMat (mainloop)
5. Menggambar objek virtual	Draw (mainloop)
6. Tutup video capture	Cleaneup

Fungsi utama dalam ArToolkit adalah *main, init, mainloop, draw* dan *cleanup*.

a. Main

Apabila kita buka file simpletest.cpp maka akan terdapat routine. Kode init diatas digunakan untuk inisialisasi video capture, membaca file *marker* , parameter kamera dan *setup graphic window* (lihat langkah satu pada tabel 3.2). Kemudian fungsi arVideoCapStart akan menghidupkan kamera. Fungsi argMainLoop yang akan melakukan *render* secara terus menerus (*looping*). Definisi fungsi argmainloop ada dalam file gsub.c.

b. Init

Rutin ini dipanggil dalam main rutin dan digunakan untuk menginisalisasi video *capture* dan membaca parameter-parameter aplikasi ArToolkit. Untuk menginisialisasi video *capture,* langkah pertama membuka path video kemudian mencari ukuran video. Kemudian diperlukan parameter yang akan digunakan dalam aplikasi ArToolkit, diantaranya adalah pola yang akan digunakan dan karakteristik kamera yang digunakan. Parameter kamera dibaca dari file data/camera_para.dat yang merupakan parameter *default*. Parameter tersebut ditransofmasikan dengan ukuran gambar yang ditangkap kamera. kemudian aplikasi parameter kamera, dan tampilan parameter tersebut pada konsol. Kemudian baca definisi pola yang digunakan, dalam hal ini pola yang digunakan adalah Hiro dari file Data/patt.hiro. Patt_id merupakan id pola *marker* yang diidentifikasi.

c. Mainloop

MainLoop merupakan rutin dimana banyak fungsi ArToolki yang di panggil dan merupakan langkah kedua sampai lima pada tabel 3.1. Langkah pertama untuk melakukan capture frame video menggunakan fungsi arVideogetImage. Frame video yang dicapture tersebut kemudian digambarkan pada layar. Kemudian fungsi arDetect*marker* digunakan untuk mencari pola yang sesuai dalam frame video yang sudah dicapture. Pola tersebut berisi variable *marker* _num dan *marker* _info yang merupakan *pointer* yang berisi informasi koordinat dan *confidence value* serta object_id setiap *marker* yang ditemukan. Sampai pada

tahap ini video *image* ditampilkan pada layar dan dianalisa sehingga frame tersebut tidak digunakan lagi sampai dapat memanggil *frame grabber* untuk memulai mengcapture *frame* video selanjutnya. Untuk cek visibilitas pola *marker* digunakan kode sperti dibawah ini. Transformasi antar *marker* dengan kamera bias ditemukan dengan menggunakan fungsi arGetTransMat. Posisi ril kamera dan orientasi relatif terhadap objek *marker* I merupakan 3x4, patt_trans.

Kemudian, objek virtual dapat digambar dengan menggunakan fungsi. Perlu diperhatikan langkah optimisasi dapat dilakukan jika tidak ada pola yang terdeteksi (k == 1), apabila sudah tidak ada pola yang terdeteksi kita dapat langsung melakukan *swap buffer* tanpa memanggil fungsi *draw*(), seprti terlihat pada fungsi dibawah ini.

d. Draw

Fungsi draw dibagi menjadi inisialisasi *rendering*, setup matrix dan *render* objek. Inisialisasi *rendering* 3D dilakukan dengan menggunakan beberapa fungsi pada ARToolKit dengan beberapa fungsi OpenGL. Yang diperlukan selanjutnya adalah mengkonversi matriks transformasi yang dihasilkan (matriks 3x4) kedalam format (array of 16 values), menggunakan fungsi `argConvGlpara`. Nilai dari array 16 tersebut berupa posisi dan orientasi dari kamera ril, sehingga dengan posisi tersebut diaplikasikan kepada kamera, virtual objek yang dirender seolah-olah posisinya berada pada *marker* . Posisi kamera virtual di set menggunakan fungsi OpenGL glLoadMatrixd(gl_para). Selanjutnya, reset OpenGL variable menjadi default.

Langkah-langkah yang dijelaskan di atas berjalan setiap waktu dalam rutin main rendering loop. Ketika program berjalan, mouse event di handle oleh fungsi mouseEvent dan *keyboard event* ditangani oleh fungsi *keyEvent*.

e. *Cleanup*

Fungsi cleanup dipanggil untuk memberhentikan *video processing* dan menutup *video path* sehingga *device* (kamera) bisa digunakan oleh aplikasi lainya.

3.2 Perancangan Sistem

Dalam pembahasan pada perancangan perangkat ini adalah sebagai berikut:

1. Perancangan *Marker*
2. Diagram *use case*
3. *calss* diagram
4. *Activity* diagram
5. *Sequence* diagram
6. *Component* diagram

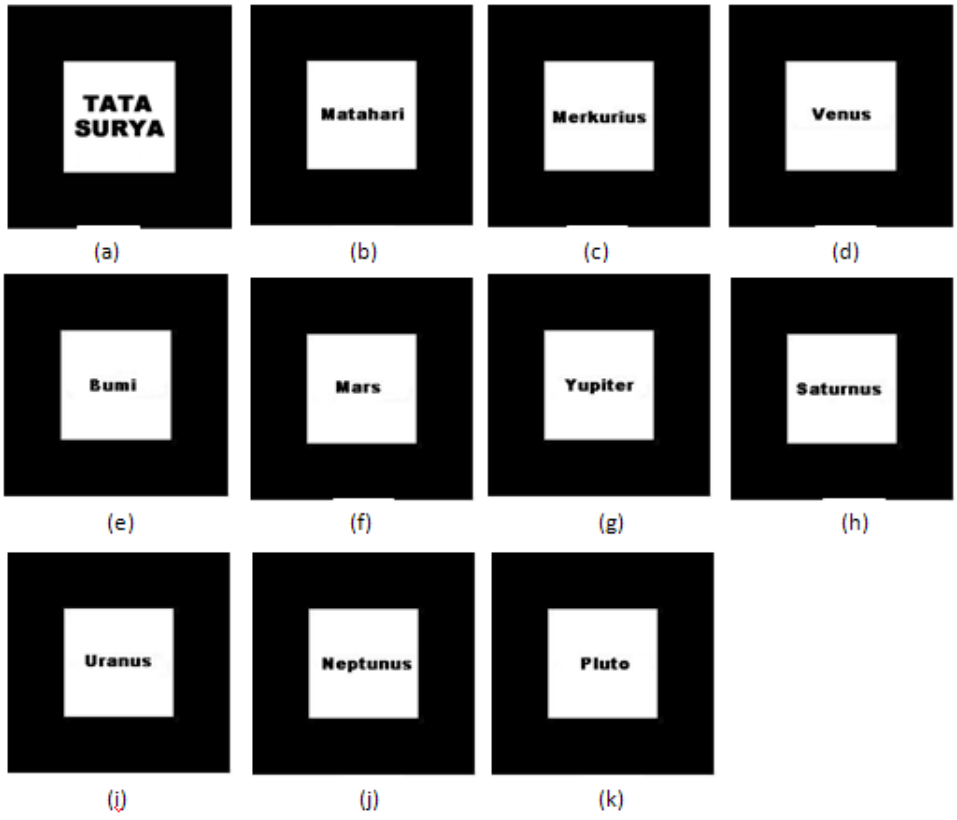
3.3.1 Perancangan *Marker*

Marker merupakan bagian yang sangat penting. Perancangan *marker* tidak boleh dilakukan sembarangan, ada aturan yang harus dipenuhi dalam merancang sebuah *marker*.

1. Dalam kasus ini marker harus berwarna hitam agar lebih mempermudah dalam proses perhitungan pendeteksian marker dan merender objek.
2. Marker yang digunakan harus berbentuk segi empat.

3. Ukuran marker akan berpengaruh terhadap objek yang akan ditampilkan. Semakin besar ukuran marker maka akan semakin jauh jarak yang dibutuhkan untuk merender objek.
4. Ketebalan marker juga sangat diperhatikan dalam membuat sebuah marker. Tebal marker disarankan minimal 25% dari panjang garis tepi marker.

   Marker-marker yang digunakan untuk menjalankan ARTatasurya ini adalah:



Gambar 3.8 Marker ARTatasurya

Keterangan:

Gambar a: marker untuk objek tatasurya

Gambar b: marker untuk objek matahari

Gambar c: marker untuk planet merkurius

Gambar d: marker untuk planet venus

Gambar e: marker untuk planet bumi

Gambar f: marker untuk planet mars

Gambar g: marker untuk planet yupiter

Gambar h: marker untuk planet saturnus

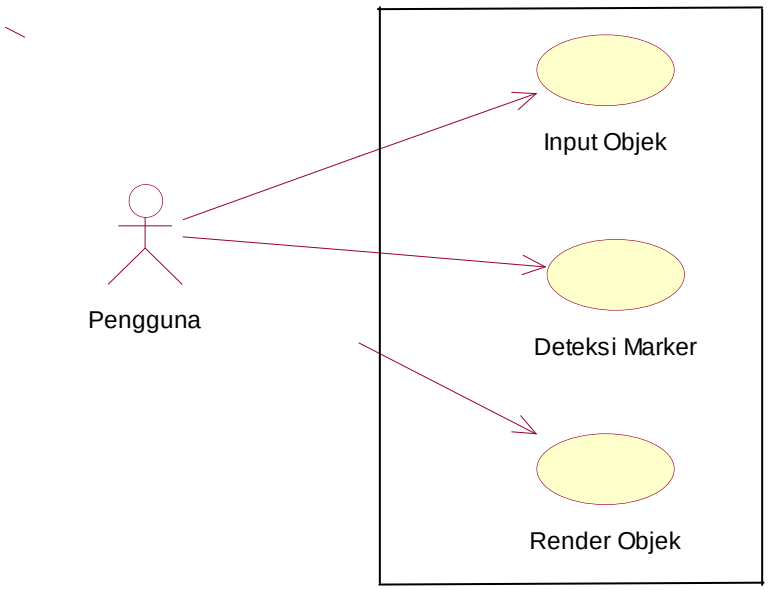
Gambar i: marker untuk planet uranus

Gambar j: marker untuk planet neptunus

Gambar k: marker untuk planet pluto

3.3.2 Diagram *Use Case*

Use Case menggambarkan fungsionalitas yang diharapkan dari sebuah sistem dan dapat merepresentasikan sebuah interksi antara actor dengan sistem.



Gambar 3.9 *Use Case* deteksi *marker*

Bagian ini menjelaskan masing-masing deskripsi dari *Use Case* diagram yang dilakukan oleh actor. Mulai dari proses input objek, proses deteksi *marker* , proses menampilkan objek, proses edit data vrml.

1) Buat Objek

Nama	: Input Objek
Aktor	: Pengguna
Deskripsi	: Export file 3D ke dalam folder wrl yang ada di ArToolkit dengan memilih type file VRML97, kemudian memberi nama objek 3D.
Kondisi sebelum	: Menu belum terbuka
Kondisi sesudah	: Tampil form ARTatasurya

Tabel 3.3 Deskripsi *Use Case* Input Objek

Aktor	Sistem
Export file wrl	
	Tidak ada
Tambah file DAT	
	Tidak ada
Buka dengan wordpad	
	Membuka wordpad
Sesuaikan file dat dengan file wrl	
	Tidak ada
Simpan File dat	
	File disimpan

2) Deteksi *Marker*

Nama : Deteksi *marker*

Aktor : Pengguna

Deskripsi : Mengeksekusi PTatasurya.exe dan tekan enter.

Kondisi sebelum : Menu belum terbuka

Kondisi sesudah : Tampil form ARTatasurya

Tabel 3.4 Deskripsi *Use Case* Deteksi *Marker*

Aktor	Sistem
Membuka file PTatasurya.exe	
	Menampilkan form utama
Klik tombol Tambah *Marker*	
	Menampilkan form parameter kamera
Tekan enter	
	Muncul Property Sheet
Tekan tombol ok	
	Menampilkan layar webcam
Mengarahkan *marker* ke webcam	
	Menandai *marker*
Klik layar webcam	
	Menampilkan form input file

	name
Ketik file name (contoh patt.xxx)	
	Menyimpan file patt

3) Render objek

Nama : Render Objek

Aktor : Pengguna

Deskripsi : Klik tombol PTatasurya.exe dan tekan enter.

Kondisi sebelum : Menu belum terbuka

Kondisi sesudah : Tampil form ARTatasurya

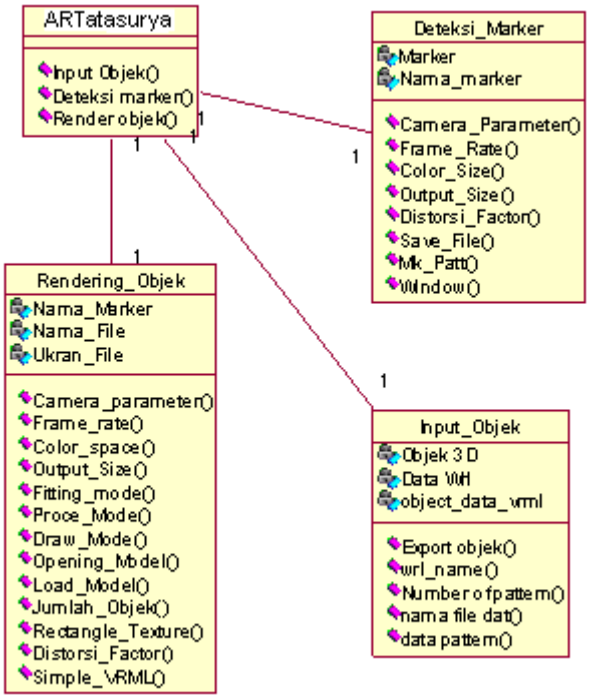
Tabel 3.5 Deskripsi *Use Case* Proses Rendering Objek

Aktor	Sistem
Klik tombol jalankan AR	
	Menampilkan Property sheet
Atur Property sheet atau langsung tekan ok	
	Menampilkan layar webcam
Arahkan *marker* ke arah webcam	
	Menampilkan objek
Melakukan transformasi	
	Melakukan pendeteksian objek
Tekan tombol keluar	
	Menutup jendela webcam

3.3.3 Class Diagram

Class diagram merupakan sebuah spesifikasi yang jika diinisialisasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangn dan desain berorientasi objek. *Calss* merupakan menggambarkan keadaan suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut. *Class*

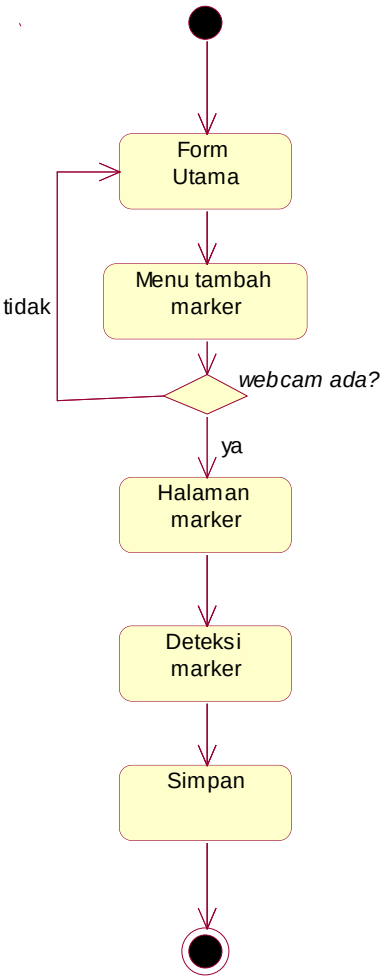
diagram menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain.



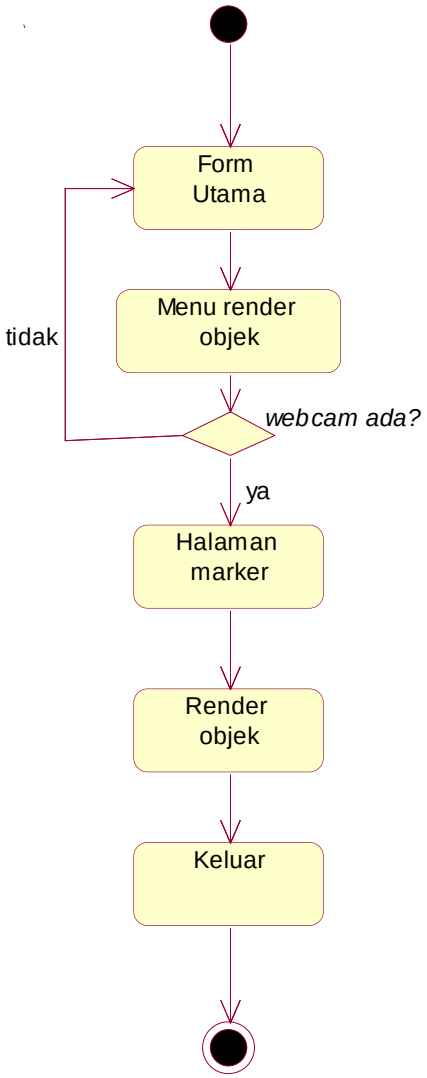
Gambar 3.10 *Class diagram*.

3.3.4 Activity Diagram

Activity diagram merupakan *state* diagram khusus, dimana sebagian besar state adalah *action*. Menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi. *Activity diagram* juga dapat menggambarkan proses parallel yang mungkin terjadi pada beberpa eksekusi.



Gambar 3.11 *Activity* diagram untuk deteksi *marker*



Gambar 3.12 *Activity* diagram untuk menampilkan objek 3D

3.3.5 Sequence Diagram

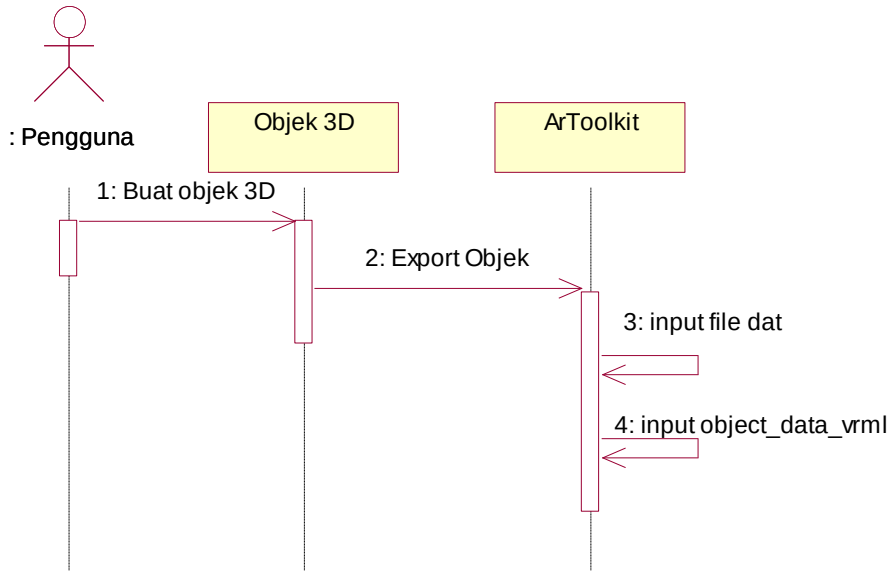
Sequence diagram menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respon dari sebuah *event* untuk menghasilkan

output tertentu. *Sequence diagram* menggambarkan interaksi antara objek di dalam dan disekitar sistem berupa pesan yang digambarkan terhadap waktu.

1) Sequence diagram input object

Scenario *sequence diagram input object:*

1. *Pengguna* mengexport file objek dari 3D max ke dalam folder ARTatasurya
2. Pilih *type file* vrml97 dan simpan.
3. Untuk memanggil file yang diexport tadi dibutuhkan file DAT. Tambahkan file dat dan edit sesuai dengan nama tipe file vrml97.
4. Edit file object_data_vrml dengan *wordpad*, samakan nama objek dat dengan file dat.

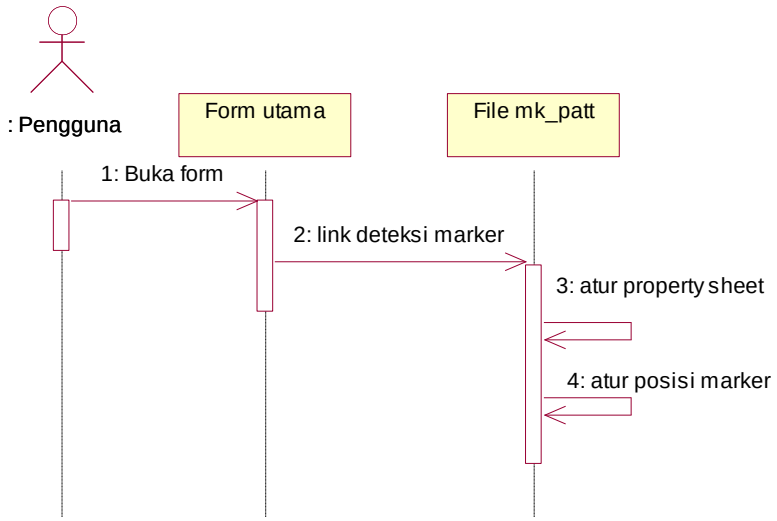


Gambar 3.13 *Sequence diagram input object*

2) Sequence diagram deteksi *marker*

Skenario *sequence diagram* deteksi *marker :*

1. Pengguna membuka *form* utama.
2. Pengguna memilih tombol deteksi *marker* untuk mendeteksi *marker* .
3. Pengguna mengatur *sheet property* yang terdiri dari *output size*, dan *color space*.
4. Layar *webcam* akan ditampilkan di *window* setelah menekan tombol ok.



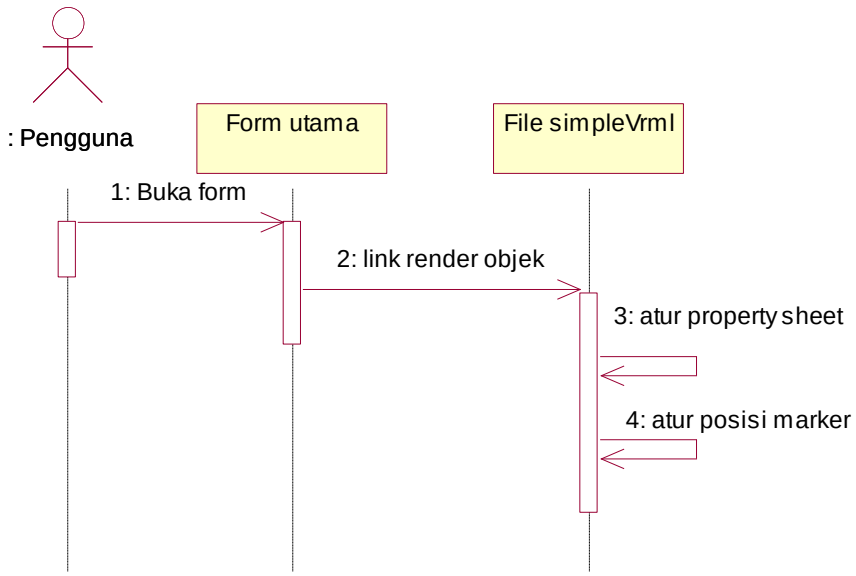
Gambar 3.14 *Sequence diagram* deteksi *marker*

3) Sequence diagram menampilkan objek 3D

Skenario *sequence* diagram untuk menampilkan objek 3D:

1. Pengguna memilih tombol Jalankan AR.
2. Seperti pada pengaturan deteksi *marker* pengguna memilih *sheet property*.

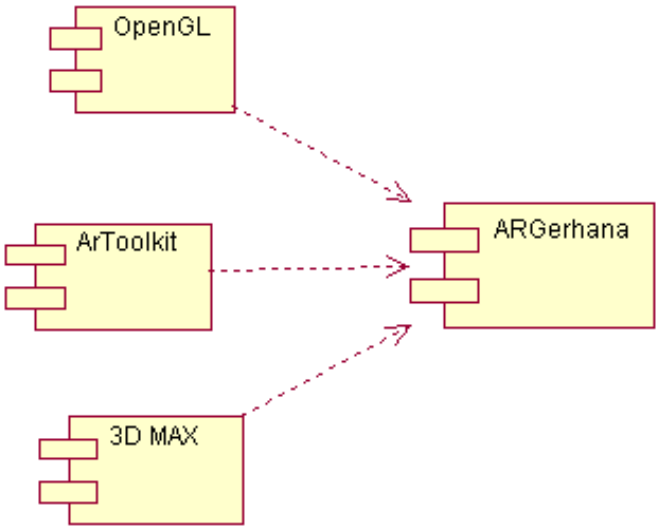
3. Layar *webcam* akan ditampilkan di *window*, pengguna harus mengarahkan *marker* kearah *webcam* dan pastikan seluruh bagian *marker* terdeteksi oleh *webcam*.



Gambar 3.15 *Sequence diagram* menampilkan objek 3D

3.3.6 Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk keteragantungan (*dependency).*



Gambar 3.16 *Component Diagram*

3.3.7 Desain Penyajian Perangkat Keras

Dalam penyajian teknologi AR, diperlukan perangkat keras yang disusun khusus agar mendapat hasil yang optimal. Komponen-komponen perangkat penyajian yang dibutuhkan adalah sebagai berikut:

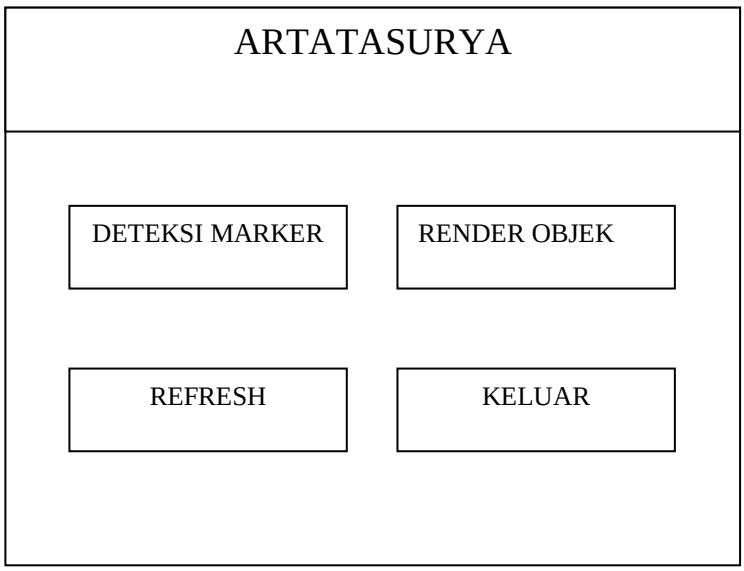
1. Meja sebagai tempat meletakan *marker* .
2. Unit visual seperti *webcam* yang mengarah ke *Marker* dengan jarak tertentu.
3. Unit proses, yang berhubungan dengan sistem komputasi dan mengendalikan jalannya sistem.
4. Unit interaksi, yang membuat pengguna dapat berinteraksi secara langsung dengan objek virtual.

Tabel 3.6 Spesifikasi desain *magibook*

No	Nama Perangkat Hardware	Ukuran	Posisi	Keterangan
1	*Marker*	631x634 pixel	Diatas meja	Kertas A4
2	*Webcam*	5 Megapixel	Diatas monitor	Micro Pack
3	Monitor	14”	Diatas meja	LCD

3.3.8 Perancangan Antarmuka

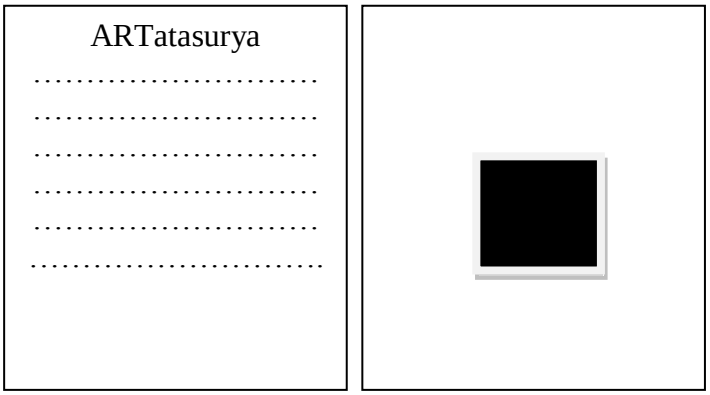
Tahap perancangan sistem dalam pembuatan suatu program merupakan suatu tahapan yang sangat penting, karena tahapan ini akan menentukan apakah suatu program akan memiliki konstruksi yang baik sesuai dengan yang diinginkan, proses pengolahan data yang tepat dan akurat serta memiliki dasar-dasar untuk pengembangan selanjutnya. Desain yang dibuat berupa *magicbook* yang mudah untuk digunakan oleh pengguna , serta informasi yang dihasilkan program tersebut dapat dimengerti oleh pengguna . Perancangan sistem bertujuan untuk memberikan gambaran sistem yang dibuat. Objek animasi tiga dimensi merupakan objek yang akan ditampilkan diatas *Marker* yang dapat dilihat oleh pengguna dengan bantuan *webcam*. Pengguna dapat melihat dari berbagai sudut pandang dengan syarat *Marker* yang ada pada magicbook harus terdeteksi oleh *webcam*. Selain itu deskripsi objek dapat dilihat pada halaman magicbook agar pengguna dapat lebih memahami objek yang ditampilkan.



Gambar 3.17 Tampilan antar muka

1. Deteksi *marker*

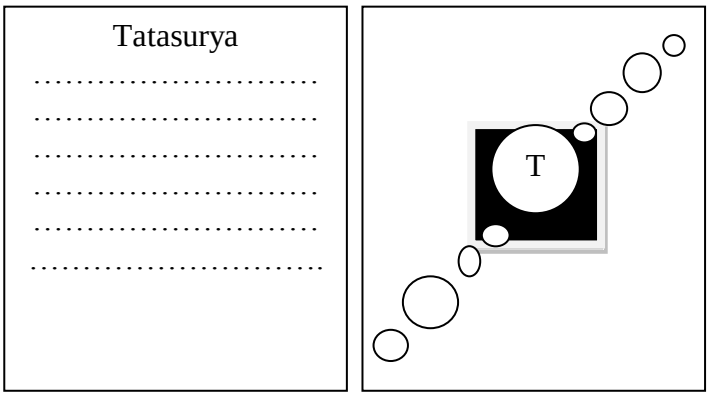
Apabila *marker* belum ditambahkan ke dalam ARTatasurya maka *marker* harus dideteksi terlebih dahulu dengan mengarahkan *marker* ke *webcam*.



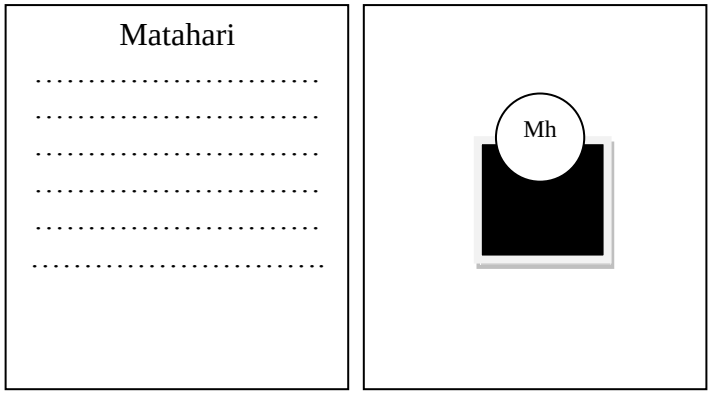
Gambar 3.18 detaksi *marker*

2. Render objek

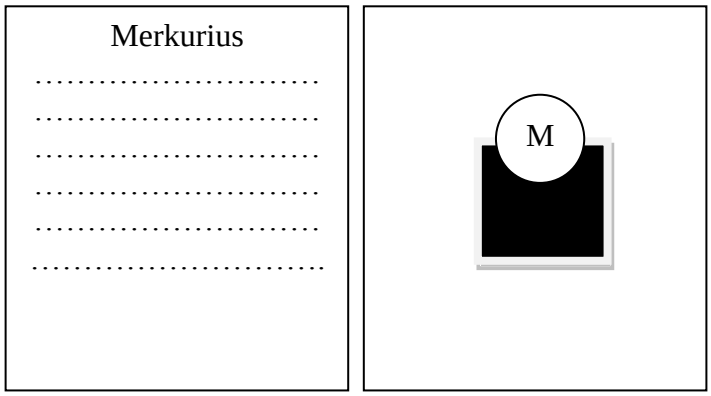
Apabila *pengguna* menekan tombol Jalankan AR dan setelah webcam mendeteksi *marker* yang pengguna arahkan ke webcam, maka akan tampil animasi 3D.



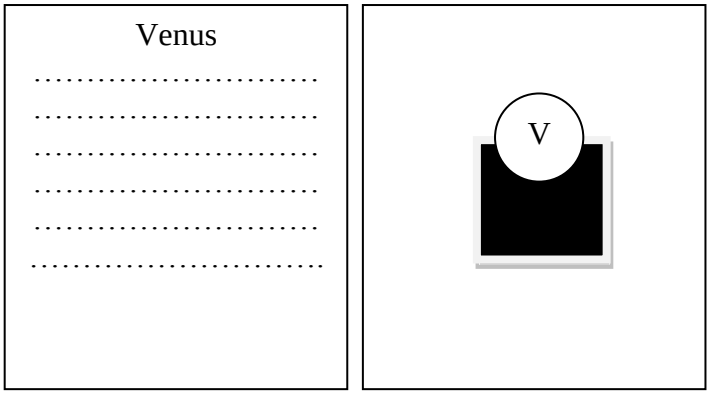
Gambar 3.19 Tata surya



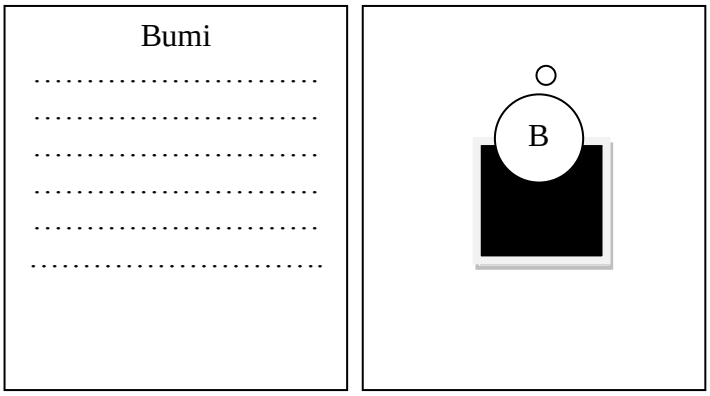
Gambar 3.19 Matahari



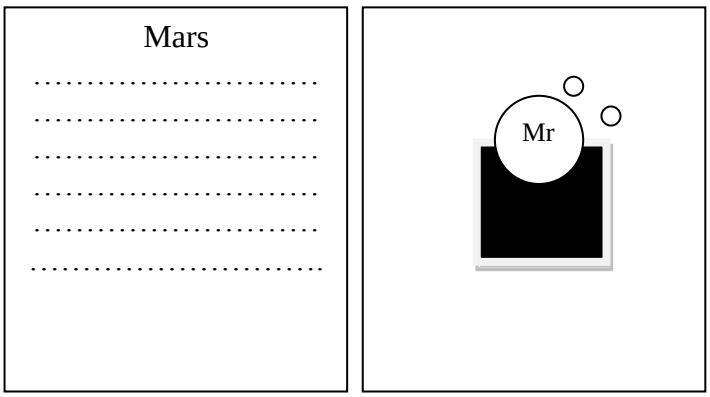
Gambar 3.20 Planet Merkurius



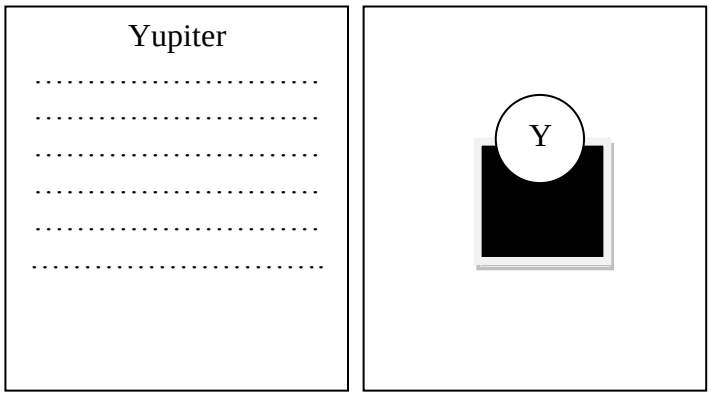
Gambar 3.21 Planet Venus



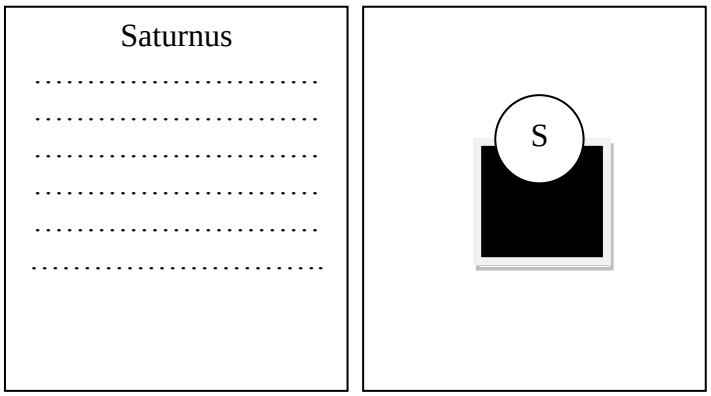
Gambar 3.22 Planet Bumi



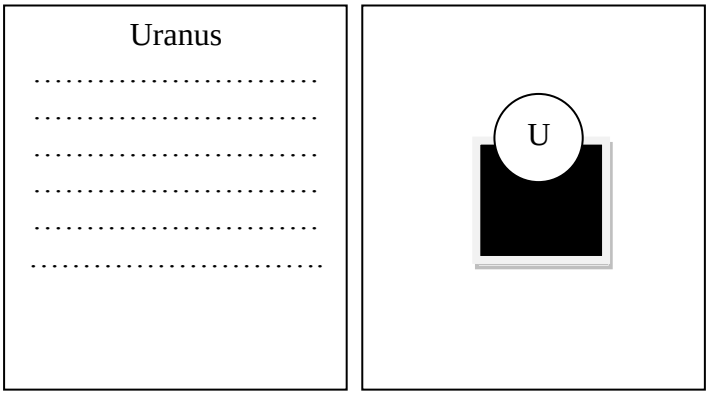
Gambar 3.22 Planet Mars



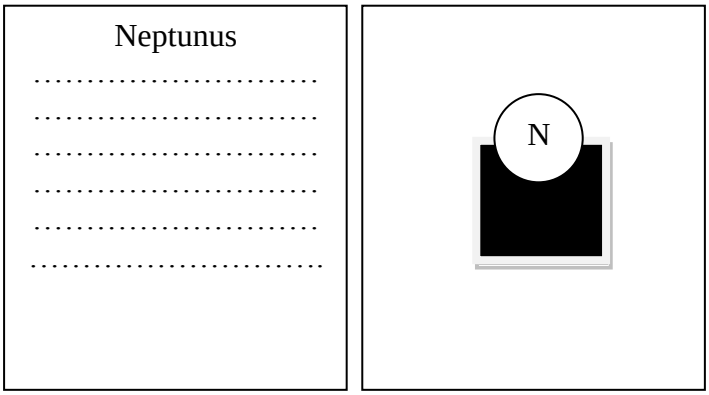
Gambar 3.23 Planet Yupiter



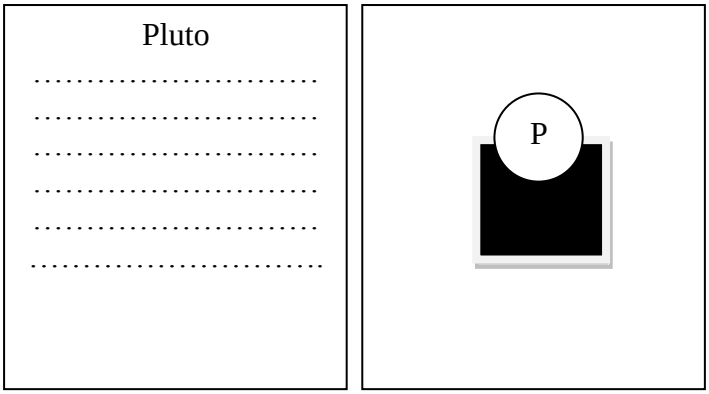
Gambar 3.24 Planet Saturnus



Gambar 3.24 Planet Uranus



Gambar 3.24 Planet Neptunus



Gambar 3.24 Planet Pluto

Keterangan:

T= Tatasurya

Mh = Matahari

M= Merkurius

V= Venus

B =Bumi

Mr= Mars

Y = Yupiter

S= Saturnus

U= Uranus

N= Neptunus

P = Pluto