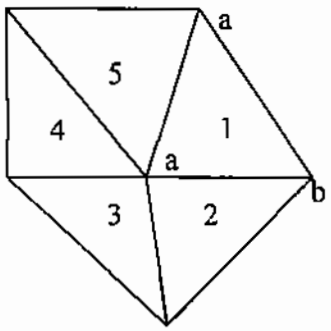
BAB III

ANALISIS DAN PEMBAHASAN

III.1. Penyusunan Index *Vertex* Dalam Algoritma

Adapun informasi dari file STL yang sedang dilakukan dalam penelitian terdiri dari array vektor segitiga dan *normal vector* nya. Setiap objek segitiga menyimpan informasi berupa nilai *normal vector* segitiga tersebut, dan index-index *vertex* penyusunnya. Dan setiap objek *vertex* menyimpan posisi *vertex* tersebut, serta *normal vector vertex* tersebut (jika ada). Penggunaan 2 buah vektor yang menyimpan objek segitiga dan *vertex*, dilakukan untuk menghindari redundansi, mengingat sebuah *vertex* dapat dimiliki lebih dari satu segitiga. Dengan digunakannya dua buah vektor yang berbeda untuk menyimpan objek segitiga dan *vertex*, maka beberapa segitiga bisa memiliki *vertex* yang sama. Berikut adalah tabel yang menggambarkan struktur data dalam penyimpanan objek *vertex* dan segitiga,



Gambar III.1. Segitiga yang saling berdekatan, 1 node bisa dimiliki 2 segitiga atau lebih.

Tabel III.1 Struktur tabel index segitiga

Index segitiga	Index *vertex* 1	Index *vertex* 2	Index *vertex* 3
1	1	46	87
2	776	456	543
3	345	523	99
4	2343	765	5678
5	8293	7921	87

Tabel III.2. Struktur Tabel Index Vertex

Index *vertex*	Koordinat x	Koordinat y	Koordinat z
1	+1.63413E-01	+9.90319E+01	-1.22082E+00
2	+0.00000E+00	+9.99999E+01	-1.00000E+00
3	+7.11646E-02	+9.84506E+01	-1.29780E+00
4	+9.98365E+01	+9.68020E-01	-1.85569E+01
5	+9.99999E+01	+9.01001E-08	-1.87777E+01
...	...	...	...

III.2. Normalisasi Sumbu

Untuk mempermudah alur perhitungan dan membuat nilai dari semua koordinat pada pada posisi normal koordinat, dengan kata lain penyesuaian letak model faset dalam ruang 3D sehingga nilai minimum dan maksimum koordinat x, y, dan z berada pada titik yang diinginkan. Agar normalisasi sumbu ini dapat dilakukan, maka nilai-nilai koordinat x minimum, x maksimum, y minimum, y maksimum, z minimum, dan z maksimum harus disimpan. Dari nilai-nilai minimum dan maksimum tersebut, dapat dibentuk sebuah *bounding box* yang membatasi model faset 3D dan membatasi daerah perhitungan (*bounded*).

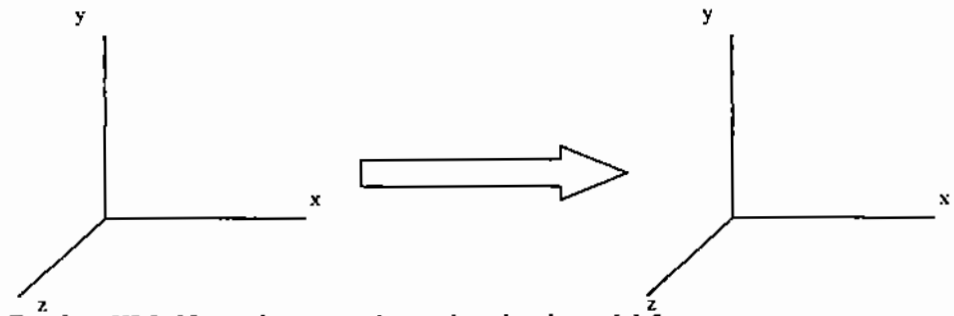
Normalisasi sumbu dilakukan dengan memindahkan bounding box, sehingga salah satu sudutnya berada pada titik pusat (0,0,0) seperti pada Gambar III.2 dan III.3. Ini berarti *vertex-vertex* penyusun model faset dipindahkan sehingga nilai dari x minimum, y minimum, dan z minimum berada pada titik (0,0,0). Pemindahan ini dilakukan dengan melakukan perhitungan berikut,
jika x minimum dilambangkan dengan xmin, y minimum dilambangkan dengan ymin, dan z minimum dilambangkan dengan zmin, maka jarak antara titik pusat dengan ketiga koordinat tersebut masing-masing adalah

$$dx = x\min - x0$$
$$dy = y\min - y0$$
$$dz = z\min - z0 \qquad \text{(III.1)}$$

Selanjutnya pada setiap *vertex* penyusun model faset dilakukan penyesuaian nilai x, y, dan z sebagai berikut,

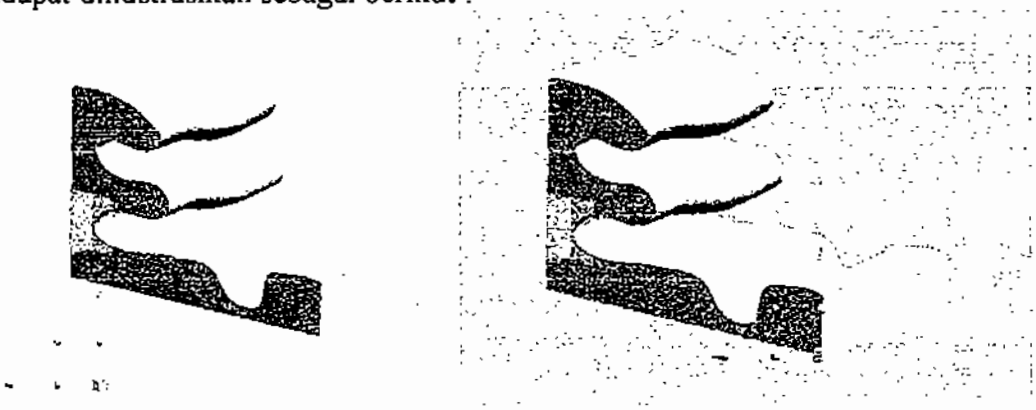
$$x' = x - dx$$
$$y' = y - dy \qquad \text{(III.2)}$$
$$z' = z - dz$$

Tujuan utama dan kegunaan normalisasi sumbu ini antara lain mempermudah proses operasi dalam pencarian *vertex* segitiga, dan mempermudah proses *pencarian segitiga bertumpuk yang memiliki 2 normal vector* selanjutnya.



Gambar III.2. Normalisasi sumbu pada sebuah model faset

Dalam struktur data obyek yang penuls teliti, normalisasi sumbu yang dilakukan didapat diilustrasikan sebagai berikut :



(a) *(b)*

Gambar III.3 a dan III.3.b. Sebelah kiri koordinat tersusun masih acak belum dilakukan normalisasi
Sebelah kanan koordinat sudah pada posisi (0.0.0) dan obyek dalam satu kuadran kerja.

III.3. *Bucket*

Fungsi *bucketing* adalah melokasikan letak segitiga berdasarkan posisinya untuk mempermudah dalam pendataan dalam array, Dalam hal ini pendeteksian dan pencarian segitiga menjadi terlokalisir. *Bucketing* dilakukan dalam bidang 2 dimensi maka model faset diproyeksikan terlebih dahulu dalam bidang 2 dimensi (x dan y) dan sumbu z sementara diabaikan. Proyeksi ini menghasilkan serangkaian segiempat yang didalamnya berisi segitiga-segitiga yang telah didata dalam masing-masing segiempat tersebut (Gambar III.4).

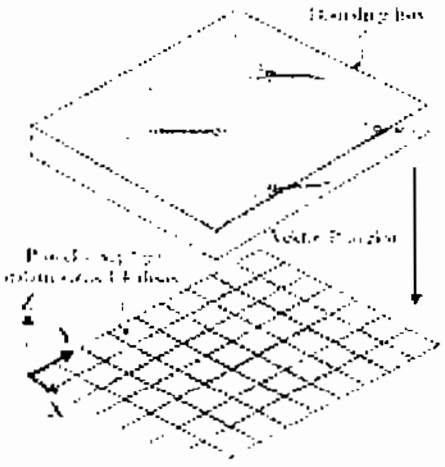
$$i = x / lebar\_bucket$$
$$j = y / tinggi\_bucket \qquad \text{(III.3)}$$

dimana i dan j merupakan jumlah indeks *bucket* yang dibuat dalam box proyeksi bidang obyek, sehingga akan didapat :

ni=delta_x/lebar_*bucket*

nj=delta_y/tinggi_*bucket*

dimana n adalah jumlah *bucket*



Gambar III.4. Proyeksi Bucketing pada sumbu X dan Y [5]

Sedangkan jumlah *bucket* yang terdapat pada kesuluruhan panjang x dan y pada bagian *feature* adalah :

$$n\_x = x\_\max / lebar\_bucket$$
$$n\_y = y\_\max / panjang\_bucket \quad \text{(III.4)}$$

Dimana :

n_x = jumlah *bucket* berdasarkan jarak antara x_min dan x_max

n_y =jumlah *bucket* berdasarkan jarak antara y_min dan y_max

x_min dan x_max merupakan nilai maksimum dan minimum *feature* tersebut.

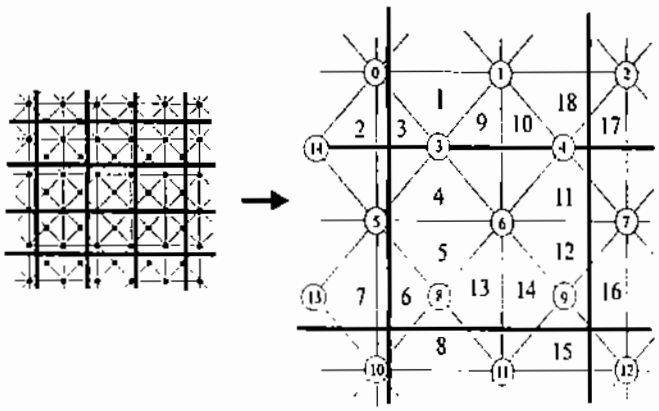
Mudah dilakukan *bucketing* bila normalisasi sumbu telah dilakukan, karena perhitungan dimulai dari (0,0). Nilai indek_*bucket* dan jumlah *bucket* akan mudah didapat bila telah dilakukan noermalisasi, namun bila normalisasi tidakk dilakukan pengurutan index perlu dicari nilai minimm dan maksimum dahulu dari *feature* tersebut pada koordinat x dan y . Dimana Rumus akan menjadi :

$$i = (x - x\_\min) / radiustool$$
$$j = (y - y\_\min) / raiustool \quad \text{(III.5)}$$

Dalam algoritmanya menjadi :

$$n\_x = (x\_\max - x\_\min) / lebar\_bucket$$
$$n\_y = (y\_\max - y\_\min) / panjang\_bucket \quad \text{(III.6)}$$

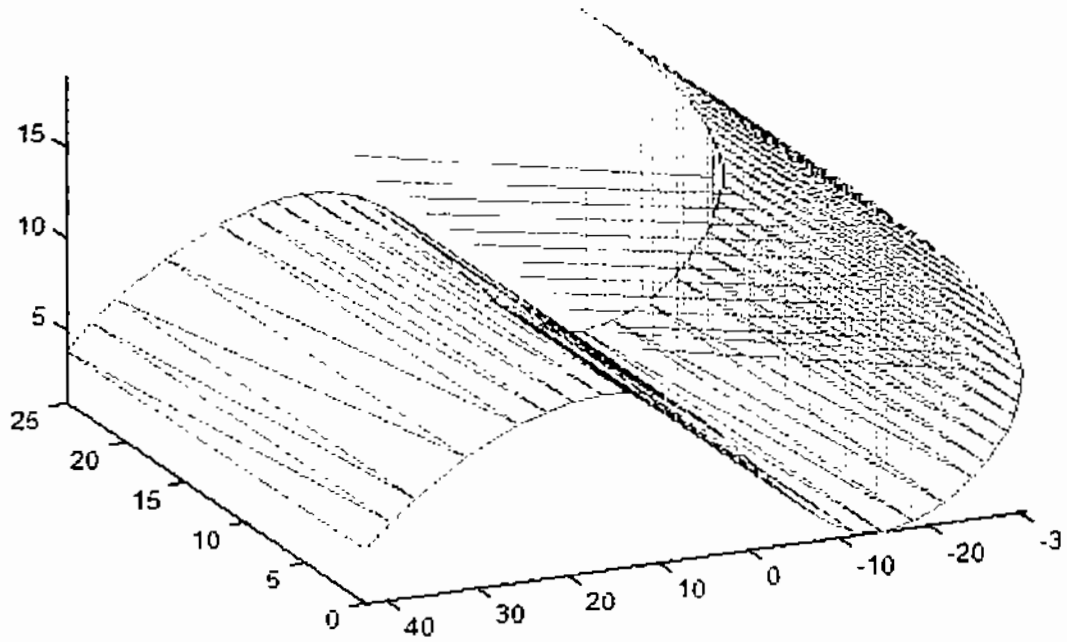
Dengan proses *bucketing* pengindentifikasi sebuah segitiga pada wilayah tertentu yang dimaksud menjadi lebih mudah dan tidak perlu membaca keseluruhan *feature*



Gambar III.5 Segitiga dalam Bucket [5]

III.4. Pembentukan Orientasi Pahat

Lintasan pahat yang dikembangkan dalam sistem ini terbagi menjadi dua, yaitu lintasan pahat untuk pemesinan awal (*roughing*), dan lintasan pahat untuk pemesinan akhir (*finishing*). Lintasan pahat ini dibentuk dari kumpulan *cc-point* (*cutter contact point*) dengan alur tertentu.



Gambar III.6. Hasil Perolehan posisi pada garis tengah lengkungan

Kehalusan yang dihasilkan dan jumlah alur mata pahat yang dibuat harus seefisien mungkin, mengingat umur mata pahat terhadap kerja dan hasil dari kehalusan *surface* yang dikerjakan, maka dibedakan menjadi 2 proses, yaitu :

III.4.1 Proses pemesinan awal (*roughing*)

Roughing atau disebut sebagai pemesinan awal adalah proses pemotongan material oleh pahat (*cutting tools*) yang bertujuan untuk menghilangkan material yang tidak diperlukan secepat mungkin hingga didapat bentuk yang mendekati bentuk akhir. Lintasan pahat untuk proses ini dibentuk dari serangkaian titik yang disebut *roughing points*, yang didapat melalui perpotongan antara bidang yang tegak lurus sumbu z, atau sejajar dengan bidang xy, dengan model faset

III.4.2 Proses pemesinan akhir (finishing)

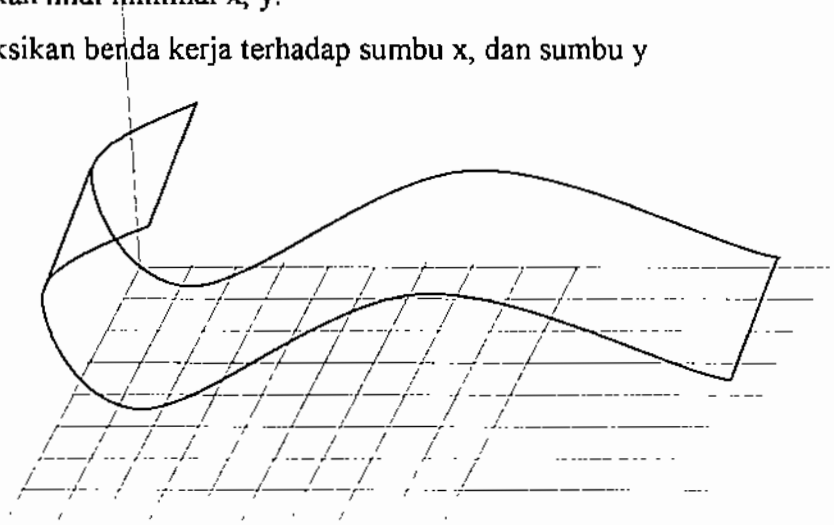
Proses *finishing* atau pemesinan akhir adalah proses pemotongan material yang bertujuan untuk mendapatkan bentuk akhir produk sesuai dengan akurasi yang telah ditentukan dan dilakukan setelah proses *roughing*. Dalam proses *finishing* ini, hal yang dilakukan adalah pembentukan *cc-point* yang merupakan lintasan pahat dengan alur tertentu dan menentukan titik *free-gouging* untuk mencegah pahat ber-*interferensi* dengan model.

Pada proses pemesinan akhir, pembentukan cc-point dilakukan dengan mencari perpotongan antara garis yang merupakan proyeksi dari lintasan pahat yang akan dibuat dengan *edge* atau *vertex* segitiga.

III.5. Langkah implementasi algoritma

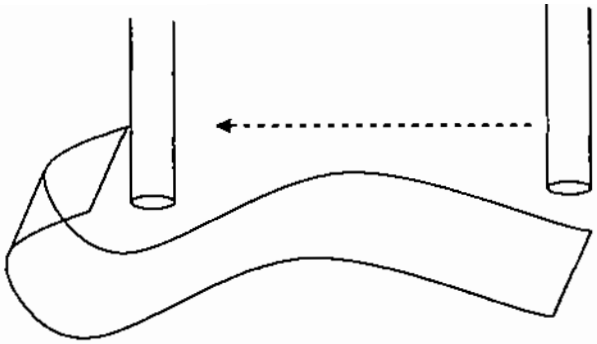
Pola alur yang dilakukan penulis untuk penyelesaian tugaas akhir ini menggunakan dasar acuan listasan pahat dan *radius tool*, pusat sumbu pahat akan menjadi dasar acuan sebagai *cutter contact point* , alurnya sebagai berikut :

1. Tentukan nilai minimal x, y.
2. Proyeksikan benda kerja terhadap sumbu x, dan sumbu y



Gambar III.7. Proyeksi terhadap bidang planar

3. Tentukan titik yang ditengarai mengandung lebih dari 1 segitiga (letaknya pada sumbu x,y) berdasar pada titik tengah pahat (nilai y tetap dan x berubah)

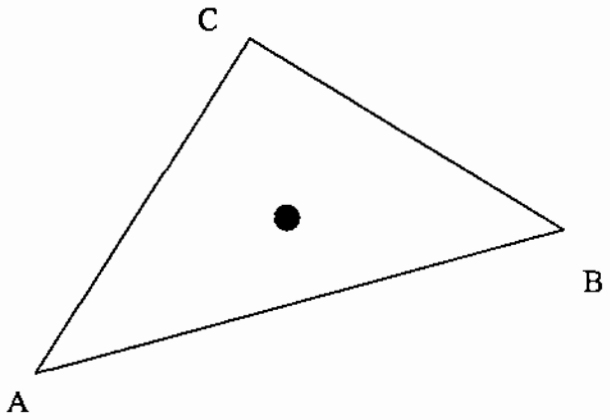


Gambar III.8. jalan alur sumbu pahat

Melakukan pembacaan titik dan mengurutkan index segitiga dengan *bucketing* dalam program Matlab dibahas dalam Bab 4

4. Pada titik tersebut cari segitiga segitiga yang berdekatan (tetangga) dengan dasar nilai *radius tool* yang ada.

Setelah segitiga-segitiga ketahui, titik tersebut harus dicek terlebih dahulu, apakah berada di dalam atau di luar segitiga. Bila berada di dalam, maka pengangkatan dilakukan sesuai dengan persamaan yang telah dibuat, namun jika berada di luar segitiga, perhitungan tidak perlu dilakukan ke langkah berikutnya.



Gambar III.9 Pengecekan apakah titik berada di dalam atau di luar segitiga

Pengecekan dilakukan dalam sistem koordinat sumbu pahat dengan memproyeksikan segitiga dalam bidang xy, dan dilakukan berlawanan arah jarum jam. Pada gambar di atas, pengecekan dilakukan pada *edge* AB, BC, dan CA secara berurutan. Persamaan garis yang menghubungkan dua titik adalah

$$fx + gy + h = 0$$

$$f = -(y_2 - y_1), g = (x_2 - x_1) \text{ dan } h = (x_1 y_2 - x_2 y_1) \qquad \text{(III.7)}$$

Nilai $t \geq 0$ bila Q_β terletak di sebelah kiri garis atau tepat pada garis. Q_β ada di dalam segitiga bila terletak di sebelah kiri semua *edge* segitiga.

Ini juga dilakukan untuk mendeteksi apakah ada segituga yang bertumpuk atau tidak, dengan mengacu pada ketentuan pembacaan dengan melihat normal vektornya.

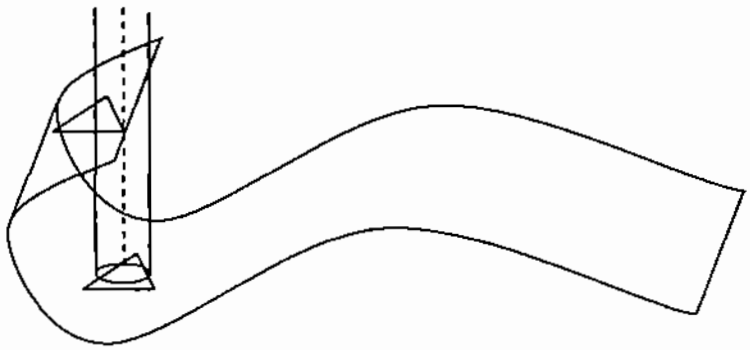
a. Untuk normal *vector* yang menghadap kebawah dilakukan dengan pembacaan : $t \leq Fx + gy + h$;

   (III.8)

b. Untuk normal *vector* yang menghadap keatas dilakukan dengan pembacaan

   : $t \geq Fx + gy + h$ (III.9)

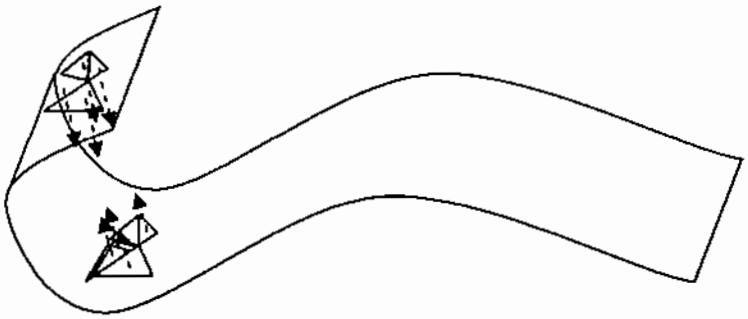
Bila kedua persamaan diatas terpenuhi dan pembacaan terhadap index *vertex* yang sudah ditentukan letak dari *radius tool*-nya, maka terdeteksi bahwa ada segitiga bertumpuk ditempat tersebut, dan diindentifikasikan lalu dicatat nilai koordinatnya.



Gambar III.10. Pendeteksian segitiga bertumpuk

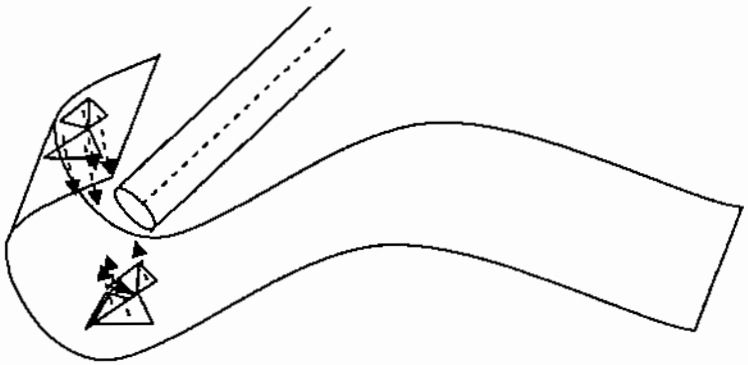
Dalam Program Matlab sebagai berikut :

5. Catat index segitiga segitiga tersebut serta tentukan *normal vector* masing masing segitiga.
6. Bandingkan nilai *normal vector* yang baru didapat dengan *normal vector* yang terdapat titik yang telah kita tentukan di atas.



Gambar III.11 Pendeteksian kesejajaran atau berpotongan pada normal vector

Jika perbedaannya mencapai >90 derajat maka diduga terdapat lipatan di daerah tersebut, dari sini baru kita bisa menentukan orientasi pahat pada bagian yang berlipat berdasarkan tinggi lipatan.



Gambar III.12 Pendeteksian posisi titik tengah diantara 2 surface lungkung

Menentukan titik tengah pada orientasi Pahat dengan menggunakan rumus :

$$\frac{\text{Nilai\_z\_segitiga\_atas Nilai\_z\_segitiga\_bawah}}{2} \qquad \text{(III.10)}$$

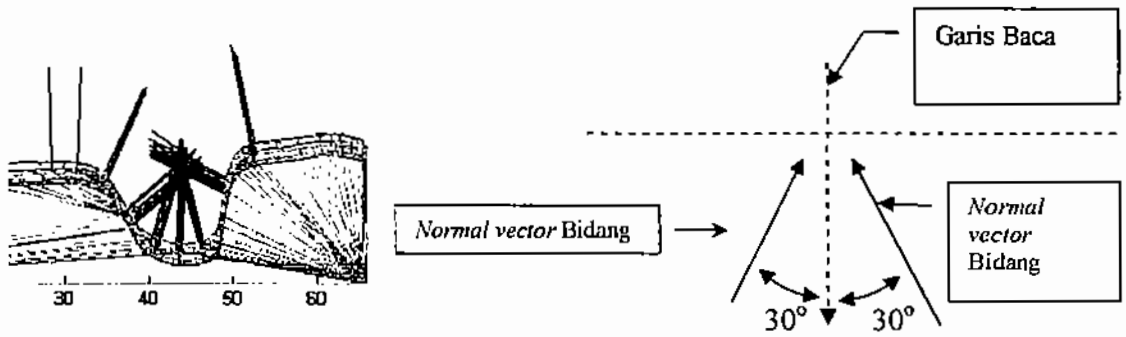
7. Langkah Selanjutnya pendeteksian *surface* dan pendataan tinggi dari nilai Z sebagai pendeteksi apakah terjadi bidang bertumpuk atau dengan kata lain identifikasi untuk *surface* yang bertumpuk.

III.6. Pengidentifikasi *Bounded volume*

Indentifikasi sebuah *bounded volume* perlu dilakukan untuk mempercepat proses analisa suatu daerah dimana pahat harus melakukan orientasi 5 axis, pengertian *bounded volume* sendiri adalah identifikasi suatu daerah dengan suatu persyaratan dimana pahat diharuskan melakukan kondisi khusus untuk memenuhi kerja dan hasil dari *surface* yang diinginkan pada daerah pengejaan tersebut. *Bounded volume* tersebut dibedakan menjadi 2 type, yaitu *Open Bounded volume* (Daerah Batas terbuka) dan *Close Bounded volume* (Daerah Batas Tertutup).

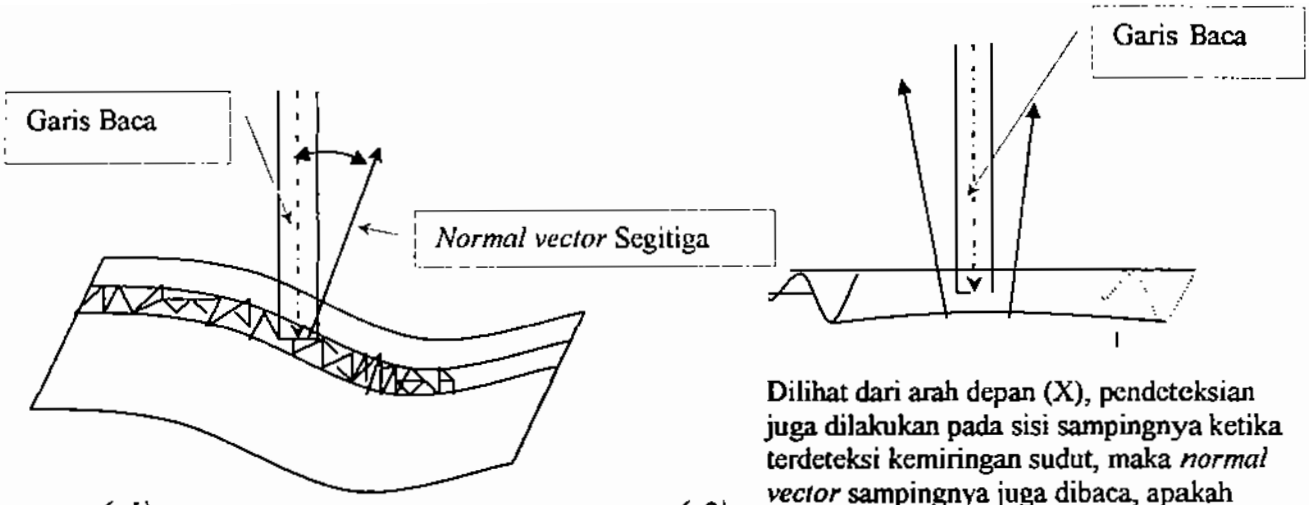
3.6.1 Daerah Batas Tebuka (*Open Bounded volume*)

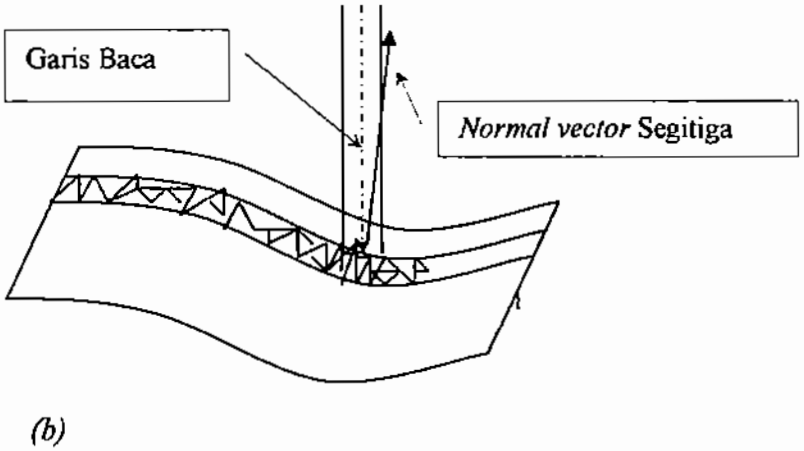
Adalah daerah dimana pengerjaan dapat dilakukan oleh pahat tanpa adanya orientasi dan deteksi awal garis sumbu pahat, dimana deteksi yang dilakuakan penulis dengan metode mengenali *normal vector* segitiga yang dilalui *radius* pahat dan dengan perbedaan derajat sebesar 30 derajat dari permukaan dan tanpa adanya segitiga bertumpuk pada *surface* kerja / *surface* yang sedang dideteksi dan dengan batasan tanpa adanya bertemu dengan bidang segitiga lainnya pada *normal vector* tersebut pada saat pendeteksian berlangsung dan adanya perbedaan tinggi. dimana diilustrasikan seperti gambar III.13 :



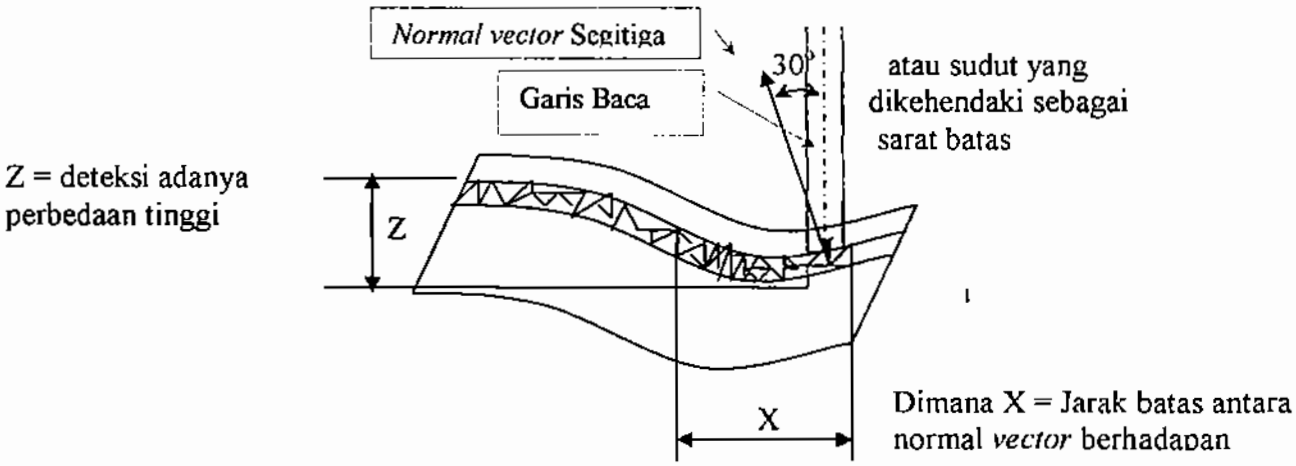
Gambar III.13 Normal vector pada bidang Open Bounded volume

Ilustrasi proses Open *Bounded volume* ditampilkan pada gambar dibawah ini, dimana diilustrasikan garis baca *normal vector* segaris dengan sumbu *tool* dan lebar daerah baca sebesar *radius tool* (Gambar III.13.a,b,c) :



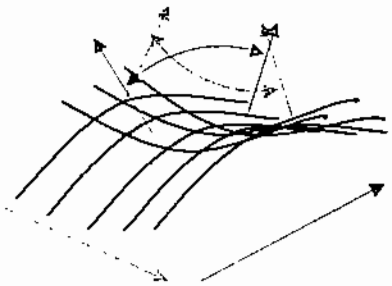


Gambar III.13,a1,a2,b Langkah Pembacaan Sumbu pada faset ketika mendeteksi Open Bounded volume dan sisi sampingnya (a2)



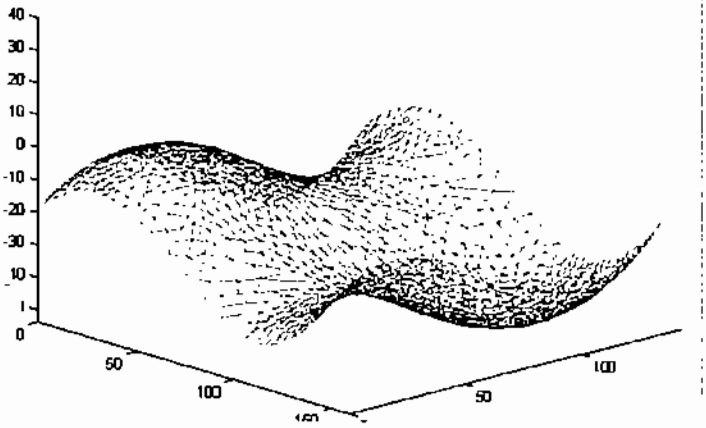
Gambar III.13.c. Langkah pembacaan lanjutan dari gambar 13.a1 dan 13.b sampai bertemu vector berlawnan sudut

Saat pembacaan Open *bounded volume* dilakukan atau dideteksi bila diketahui ada perbedaan garis pembacaan berbanding dengan normal *vector* segitiga. Bila diketahui ada bidang dengan kecekungan tetapi disisi sampingnya terjadi kemiringin berlawanan maka dideteksi sebagai *sadel point* seperti dijelaskan pada gambar dibawah ini (III.13.d,e)



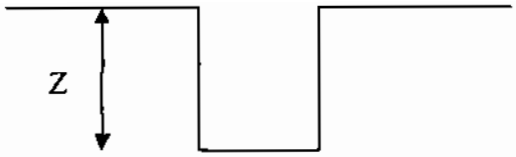
Kesadelan terjadi pada bidang cekung (garis vektor hijau) tetapi sisi samping menurun (garis vektor merah)/(bentuk

(d)



(e)

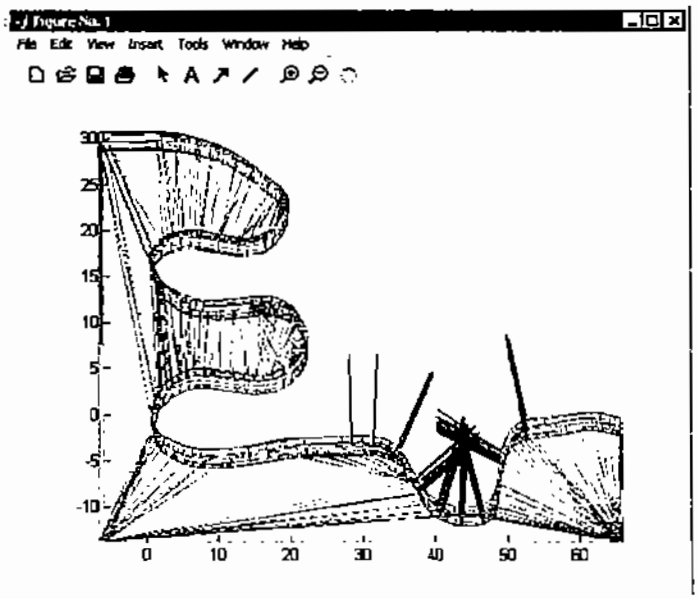
Gambar III.13.d,e Contoh Ilustrasi pembacaan bidang kesadelan dalam Program



(a)

Gambar III.14.a Bila tidak ada perbedaan sumbu maka dideteksi perbedaan tingginya

(b)

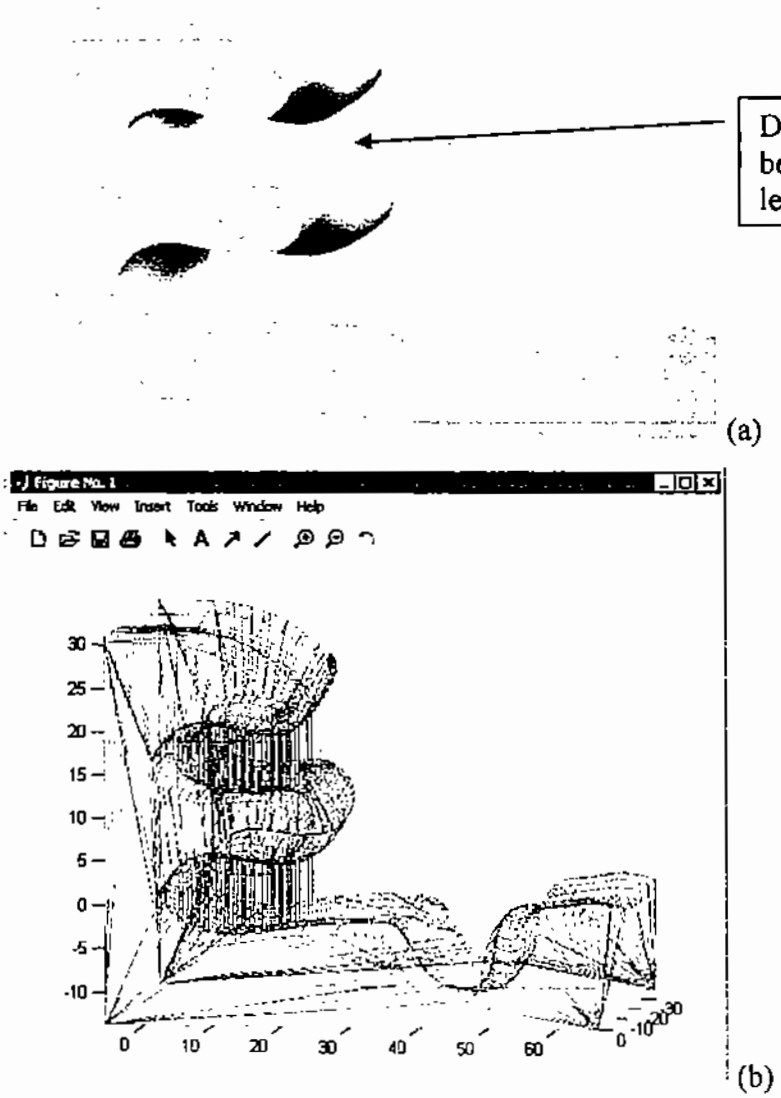


(c)

gambar III.14.b.c. Deteksi Simpangan derajat antara normal vector dan sumbu pahat (segaris dengan garis pembacaan) pada OBV.

3.6.2 Daerah Batas Tertutup (*Close Bounded volume*)

Daerah batas tertutup adalah daerah dimana memiliki keterbatasan ruang gerak pahat yang dapat diaplikasikan dalam daerah tersebut, dalam daerah ini biasanya pahat harus melakukan orientasi pemesinan 5 axis untuk dapat menjangkau kerja pada daerah tersebut, pendeteksian adanya daerah batas tertutup dilakukan pada proses di Sub-bab III.5, dimana pendeteksian semua *normal vector* , indeks segitiga dan segitiga bertumpuk. Penandaan daerah tersebut dengan adanya segitiga bertumpuk pada saat pengindentifikasi alur pahat, diilustrasikan penulis pada gambar dibawah ini :



Gambar III.15 a,b Deteksi segitiga bertumpuk pada CBV.

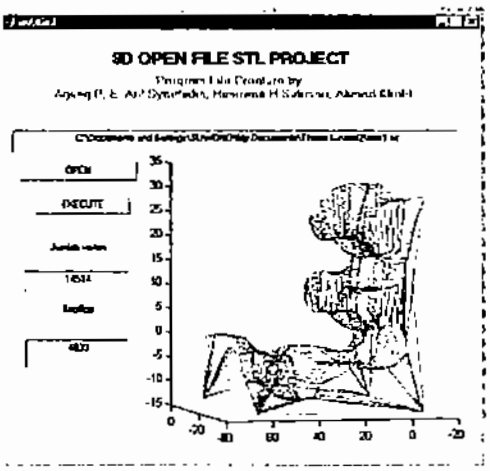
BAB IV

IMPLEMENTASI ALUR AL-GORITMA PEMOGRAMAN DALAM *SOFTWARE MATRIK LABORATORY*

Implementasi al-goritma deteksi alur pahat yang dilakukan pada bidang kerja adalah sebagai berikut :

IV.1 Penentuan Countur Obyek Awal berdasarkan STL file

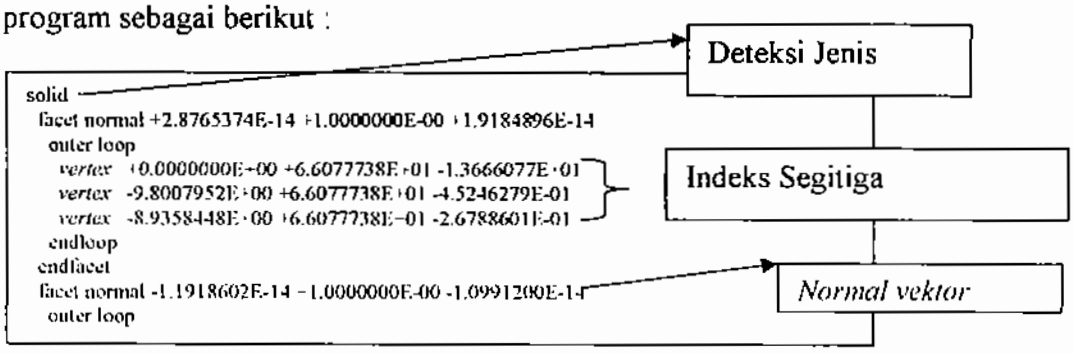
Penetuan *counter* obyek awal berdasarkan STL file dengan koordinat yang terkandung didalamnya bila diimplementasikan dengan bidang 3 dimensi pada sebuah grafik akan memenuhi menjadi :



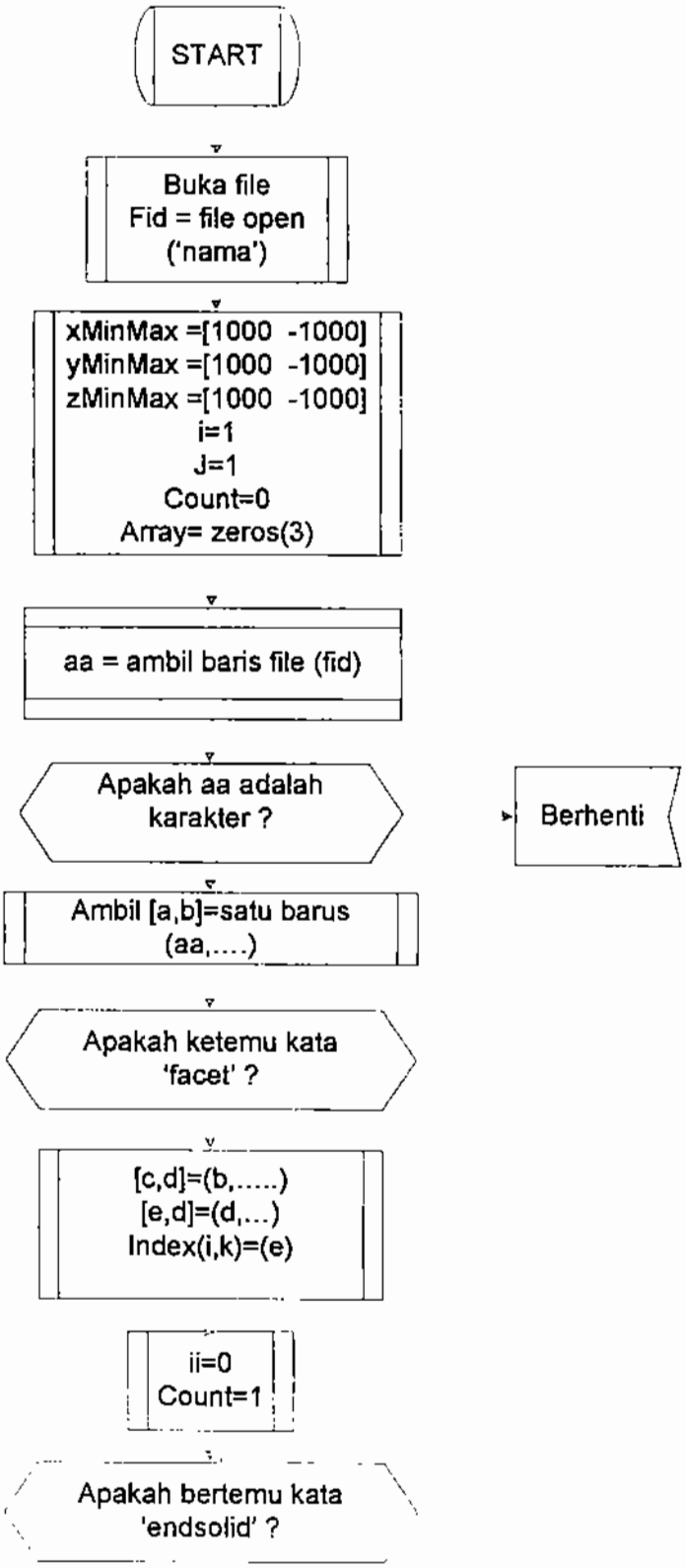
Gambar IV.1. Bentuk Solid model dalam obyek 3 Dimensi

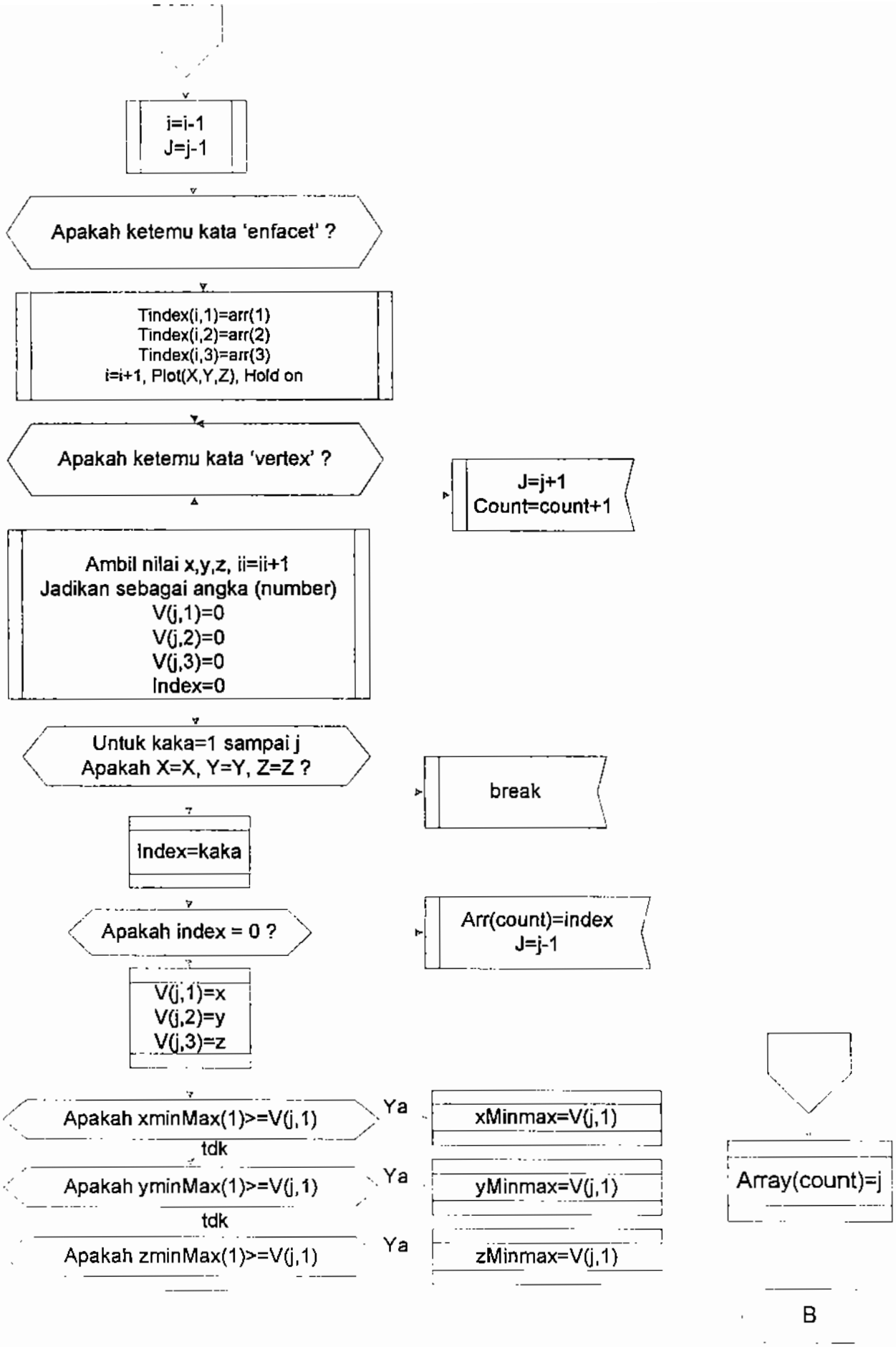
Dari informasi dalam sebuah file stl yang dibaca dengan bentuk solid, Selanjutnya membaca indek *vertex* dan memasukkannya kedalam index urut serta mengeliminasi *vertex* yang telah ada dalam index urut, dengan diilustrasikan kedalam program sebagai berikut :

:



Dimana :
Benda alur algoritma diatas dijelaskan dalam flow chsrt seperti dibawah ini :



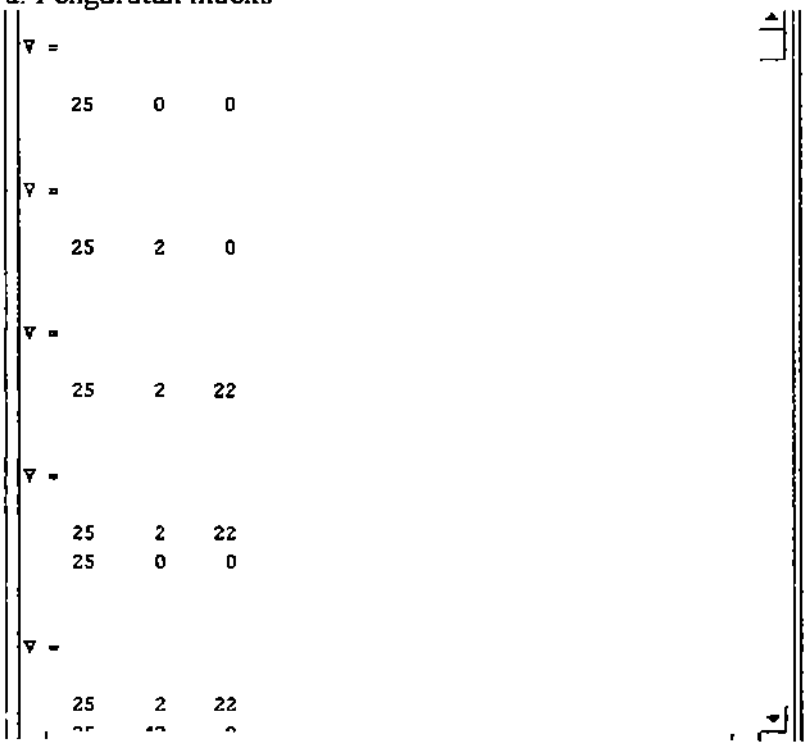


Dalam MATLAB pemograman yang ditulis adalah sebagai berikut :

```
fid=fopen('eko.txt');          % Awal file bejalan dan mendeteksi sebuah file yang dimuat ... WHI...
  ... DO TO MAKE 3D STL FILE
xMinMax = [1000 -1000];        % tolong sediakan nilai koordinat bawah 1000 nilai koordinat atas ...
yMinMax = [1000 -1000];        % begini bila ada nilai 1000 lalu dimasukkan nilai 30 mana yang ...
zMinMax = [1000 -1000];        % yang paling kecil adalah 30 karena dicatat nilai sebelumnya 1000
i=1                            % persiapan untuk counter perhitungan, hampir seluruh program dengan ...
jk=0;
arr = zeros(3);
disp ('Please waiting until all command succesfully to do and show to you the graphic of object stl model')
while (1);                     % Nah disini bilang SAAT... bla bla bla... do..., saat mengacu pada ...
  aa=fgetl(fid);               % command pertama, ambil nilai aa pada baris dari file fid
  if ~ischar(aa),              % jika bukan character text, lakukakan....
    break;                     % tutup !!!
  end                          % selesai.... programnya brenti ga mau jalan.
  [a,b] = strtok(aa,' ');      % command kedua : ambil string a dan b adalah kata pertama dari aa dan ...
  if strcmp(a,'facet'),        % command ketiga : IF (jika) setelah compare sama, kompare sini pada ...
    [c,d]= strtok(b,' ');      % command keempat : Untuk keperluan matrik c ambil dari kalimat string ...
    for k=4:6,                 % UNTUK (FOR) command ke 4 sampai seterusnya... lakukan
      [e,d] = strtok(d,' ');   % ambil nilai e,d,e
      R1 = inline(e);          % masukkan nilainya sebagai variable R1
      T(i,k) = R1(1);          % pada Indeks T(i,k), maksudnya pada indeks yang ke nilai i ...
    end
    ii = 0;
  elseif strcmp(a,'endsolid'),
    disp                       ('jumlah segitiga pembentuk obyek adalah :');
    i=i-1                      % counter loop for segitiga bila ketemu end solid dikurangi 1, kenapa ...
    disp ('jumlah total vertex');
    vertex=jk
  elseif strcmp(a,'endfacet'), % bila bertemu ENDFASET catet tuh counternya...
    i=i+1;                     % pencatat counter polygon (for model faset is given name a triangle ...
    plot3 (X,Y,Z,color)        % untuk setiap vertex yang ketemu, dimdelx... DI-PLOT-KAN !!!!
    hold on;                   % Dipertahankan gambar grafiknya walau command yang baru masuk
  elseif strcmp(a,'vertex'),   % pertama, bila ketemu vertex ------ LAKUKAN, !!!! catat dan lanjutkan ...
    ii=ii+1;                   % indexing counter <--- counter untuk index vertex
    [a,b]= strtok(b,' ');      % langkah kedua (string pertama masuk point matrik a, lainnya masuk ...
    [b,c]= strtok(b,' ');      % langkah ketiga (sama dengan berikutnya...)
    R2=str2double(a);          % untuk command lainnya bisa diganti dengan inline (dimana string yang ...
    X(ii) = R2(1);             % pencatat pada posisi X yang ke ii (index)
    R2=str2double(b);          % pencatat string dan melanjutkannya merubah string menjadi character ...
    Y(ii) = R2(1);             % pencatat pada posisi Y yang ke ii (index)
    R2= str2double (c);        % command ini sama dengan perintah sebelumnya, untuk mengambil nilai ...
    Z(ii) = R2(1);             % pencatat pada posisi Z yang ke ii (index)
    jk=jk+1;                   % counter penghitung jumlah vertex yang dikerjakan.
  end
```

Hasil dalam program adalah :

a. Pengurutan indeks



b. normalisasi untuk mendapatkan koordinat posisi (0,0,0),memudahkan perhitungan.

Sebelum Normalisasi :

V =

x	y	z
25	2	22
25	42	22
0	42	22
0	2	22
25	42	2
0	42	2
25	2	2
0	2	2

Sesudah normalisasi :

V =

25	0	0
25	40	0
0	40	0
0	0	0
25	40	-20
0	40	-20
25	0	-20
0	0	-20

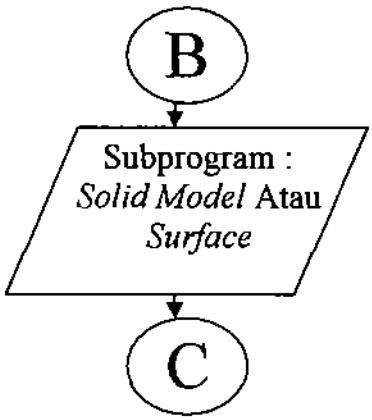
IV.2. Operasi Inti

4.2.1. Pembacaan *Normal Vector* dengan *Radius Tool* dan Mendeteksi Kesolidan

Metode untuk menampilkan ruang *bounded volume* dan membedakan *counter* permukaan dengan menggunakan *radius tool* untuk membaca indeks segitiga yang dilalui kemudian mencatat nilai *normal vector* nya, yang kemudian dibandingkan dengan toleransi arahnya, apakah sarat dari sub sebelumnya *Solid* atau *Surface*?, bila *surface* pembacaan *normal vector* hanya dilakukan satu arah dan bila solid maka pembacaan *normal vector* untuk setiap *vertex* dilakukan.

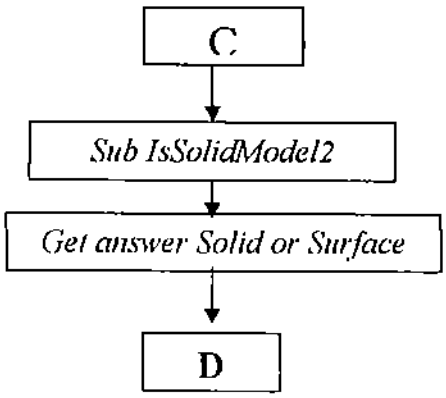
Berikut alur diagram-(*flowchart program*)-nya:

Menegecek benda apakah Solid atau bukan Solid




```
"Cek apakah bentuk yang bersangkutan adalah solid?"
disp('apakah solid??');
"Panggil isSolidModel2(V,T,xMinMax,buket,isiBuket,lebarBucket,jmlBuketY,jmlSegitiga)"
kesolidan = isSolidModel2(V,T,xMinMax,buket,isiBuket,lebarBucket,jmlBuketY,jmlSegitiga);
if kesolidan==1
   disp('ya solid');
else
   disp ('surface');
end
```

dimana Subprogram alur pembacaan Solid Model SolidModel adalah :

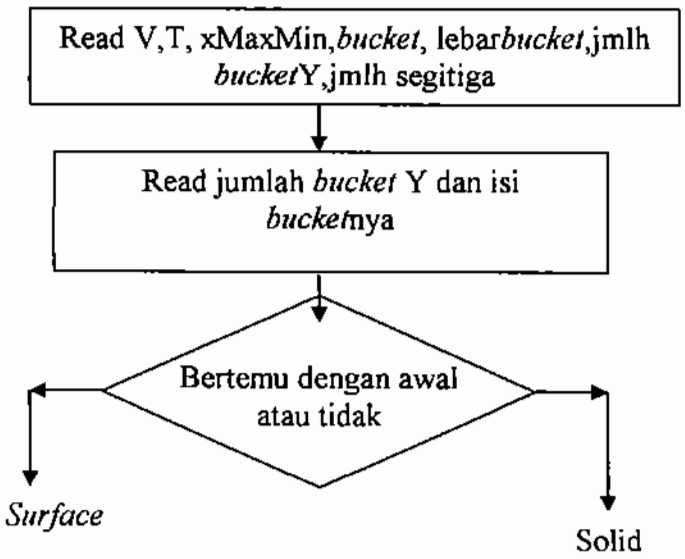


```
kesolidan                                                    =
isSolidModel2(V,T,xMinMax,buket,isiBuket,lebarBucket,jmlBuketY,jmlSegitiga
),
if kesolidan==1,
   disp('ya solid'),
else
   disp('Bukan, hanya Surface');
end
```

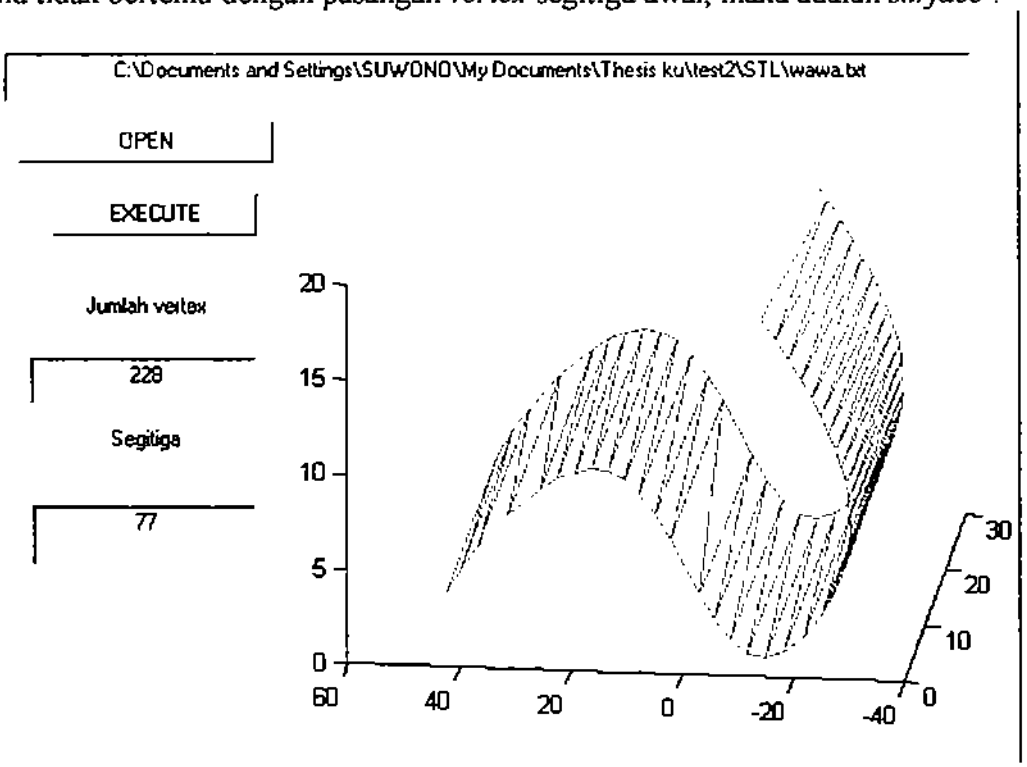
Dimana Main program akan memanggil Sub Program IsSolidModel yaitu berupa fungsi A=isSolidModel2, yang isinya :

a. Menentukan variable nilai pembacaan, disini veriale ditentukan dengan nama xRandom
b. Deteksi nilai erdekatan dengan posisi yang diacu pada sumbu x
c. Deteksi polygon (segitiga) dengan pasangan *bucket*nya.
d. Membaca dua *vertex* diawal dan menyimpannya untuk kemudian dibandingkan pada dengan nilai pasangan pembacaan berikutnya pada pengambilan nilai random (acak pada salahsatu arah x atau y)

Digambarkan dalam *flowchart* pemograman sebagai berikut :

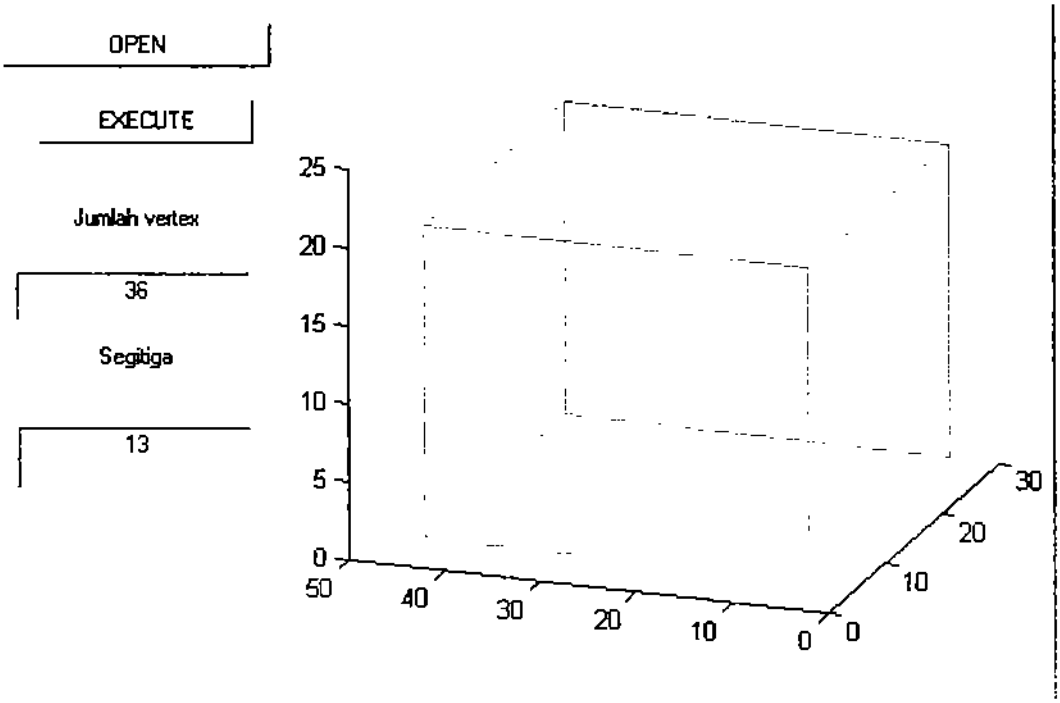


Ilustrasi dalam model :
Bila tidak bertemu dengan pasangan *vertex* segitiga awal, maka adalah *surface* :



Gambar IV.2 Contoh Bentuk Deteksi Surface

Bila terdeteksi pasangan vertek segitiga bertemu dengan awal, maka deteksi adalah Solid

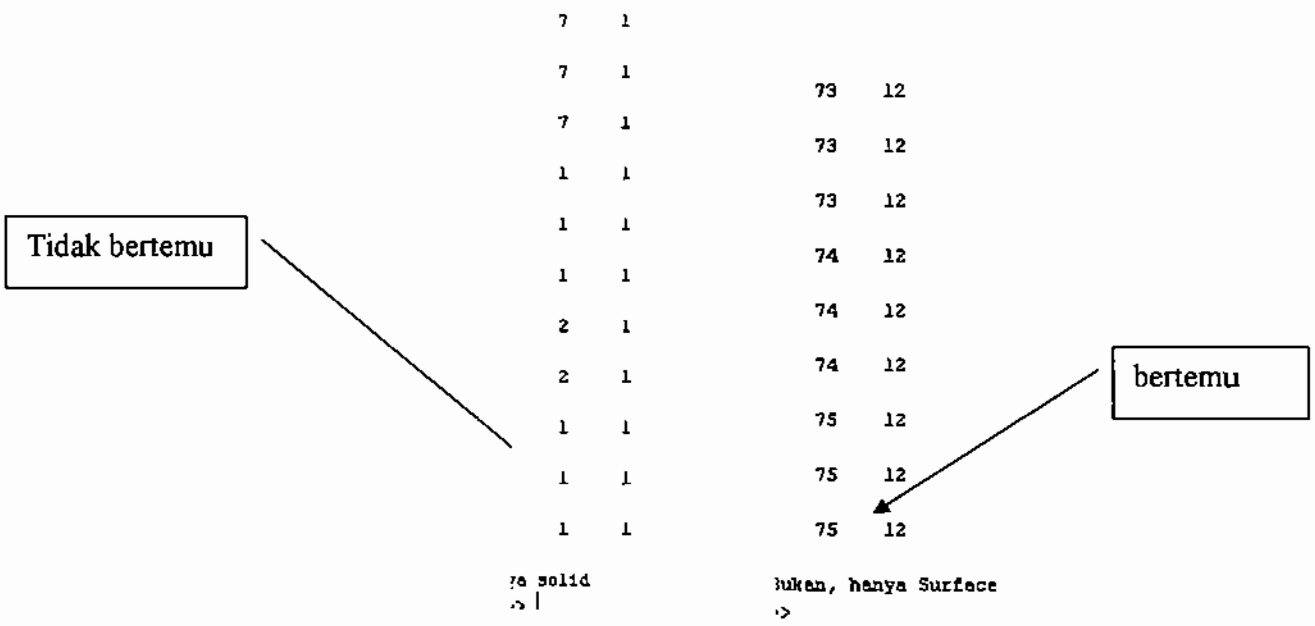


Gambar IV.3 Contoh Bentuk Deteksi Solid

Setelah terdeteksi dikembalikan lagi ke awal .

Dalam program hasilnya adalah :

bila *surface* terjadi, indek segitiga awal akan bertemu dengan salah satu segitiga yang dibaca :

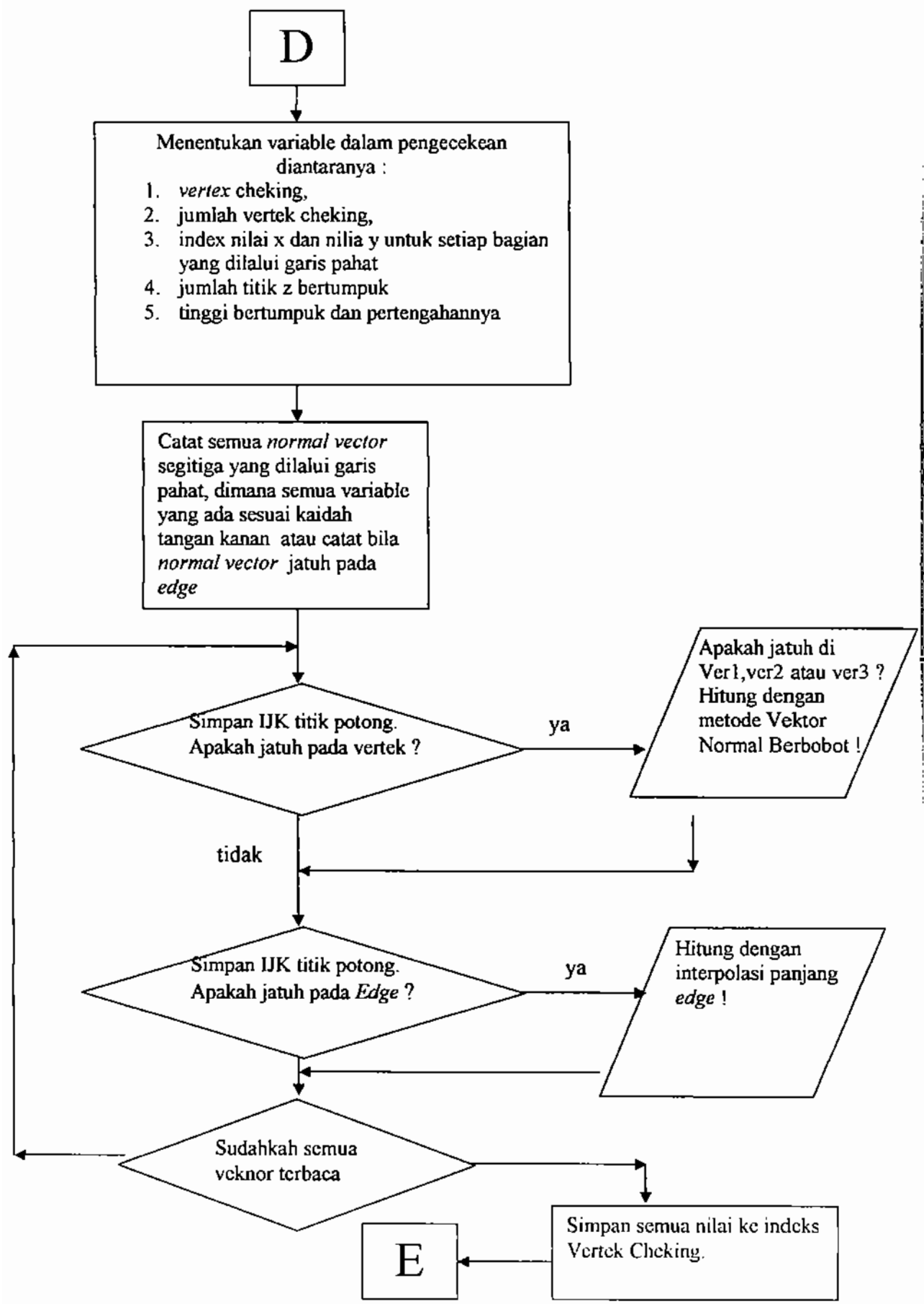


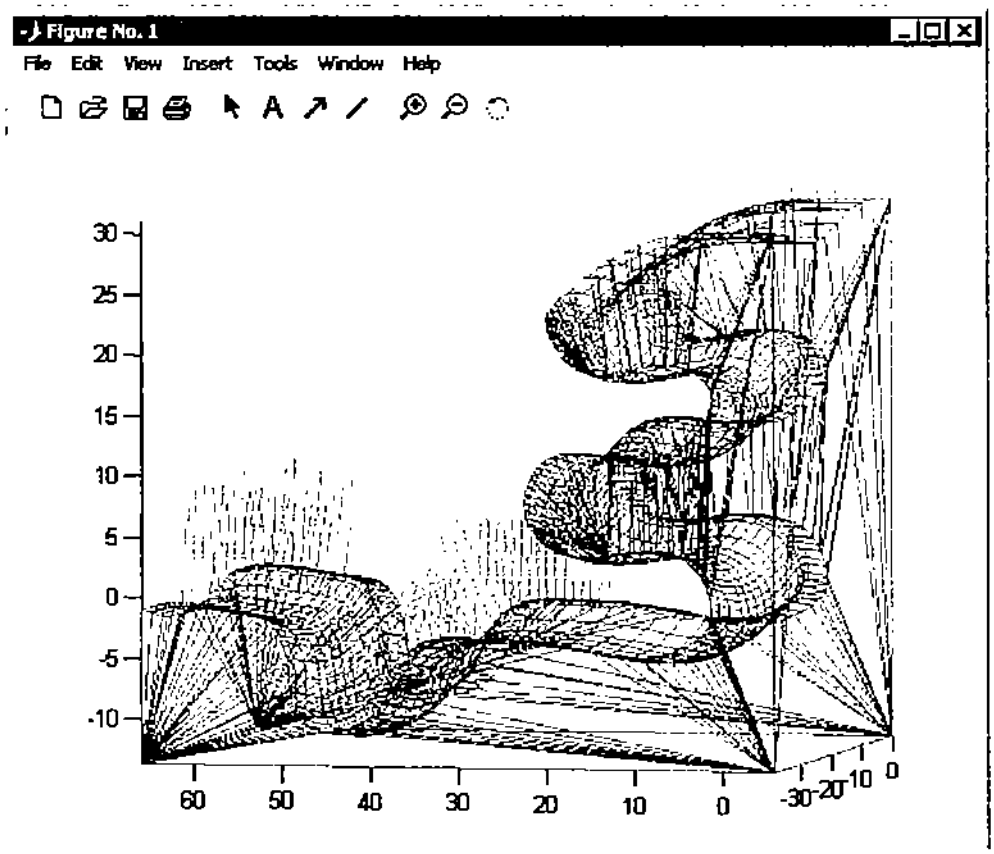
4.2.2. Mendeteksi Ruang terbatas dan Menampilkannya

Langkah program selanjutnya adalah membedakan ruang terbatas dan *menampilkannya*

List program terlampir dalam lapiran (tidak dijabarkan disini)

Flowchart yang dijalankan pada langkah ini adalah :

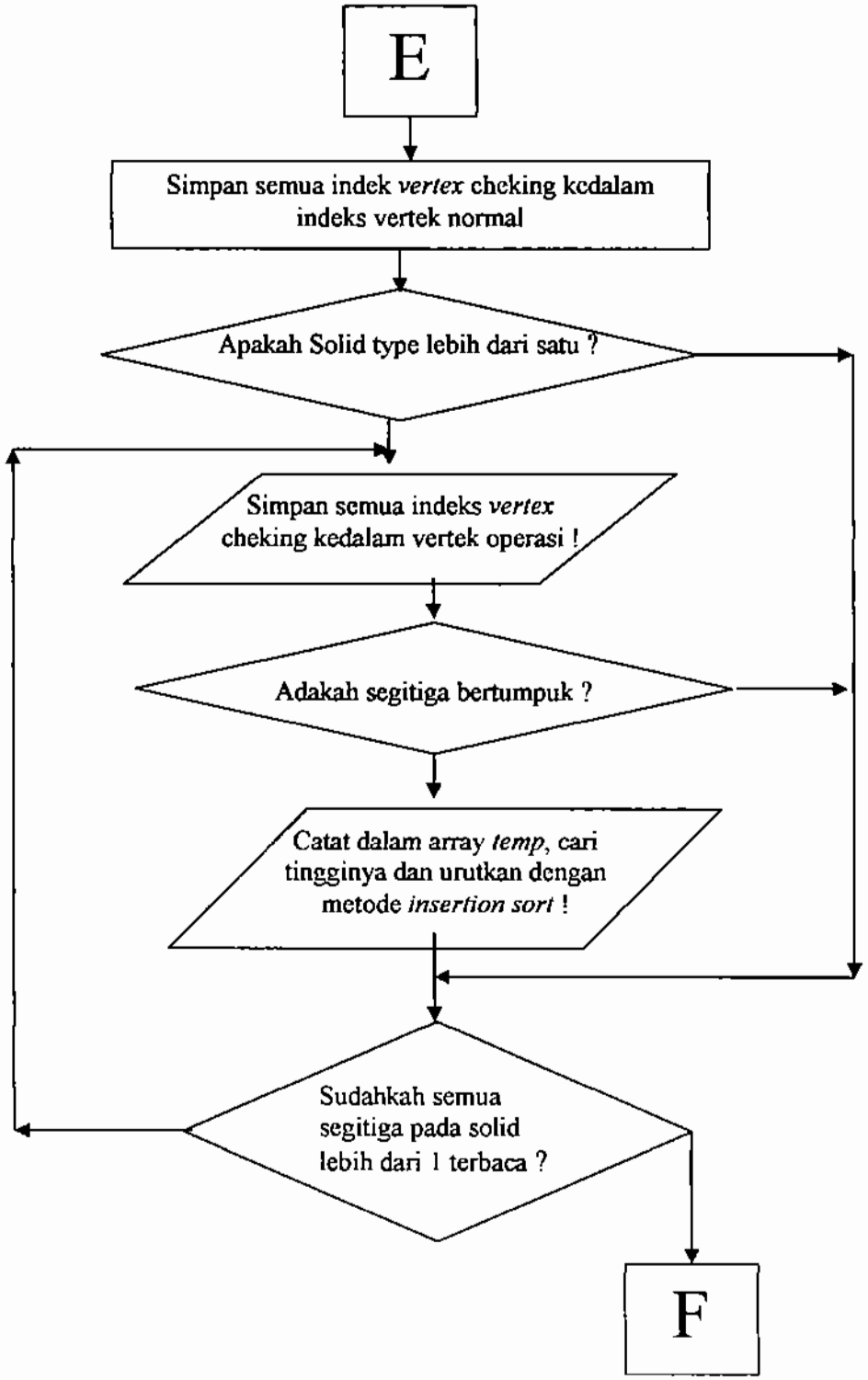


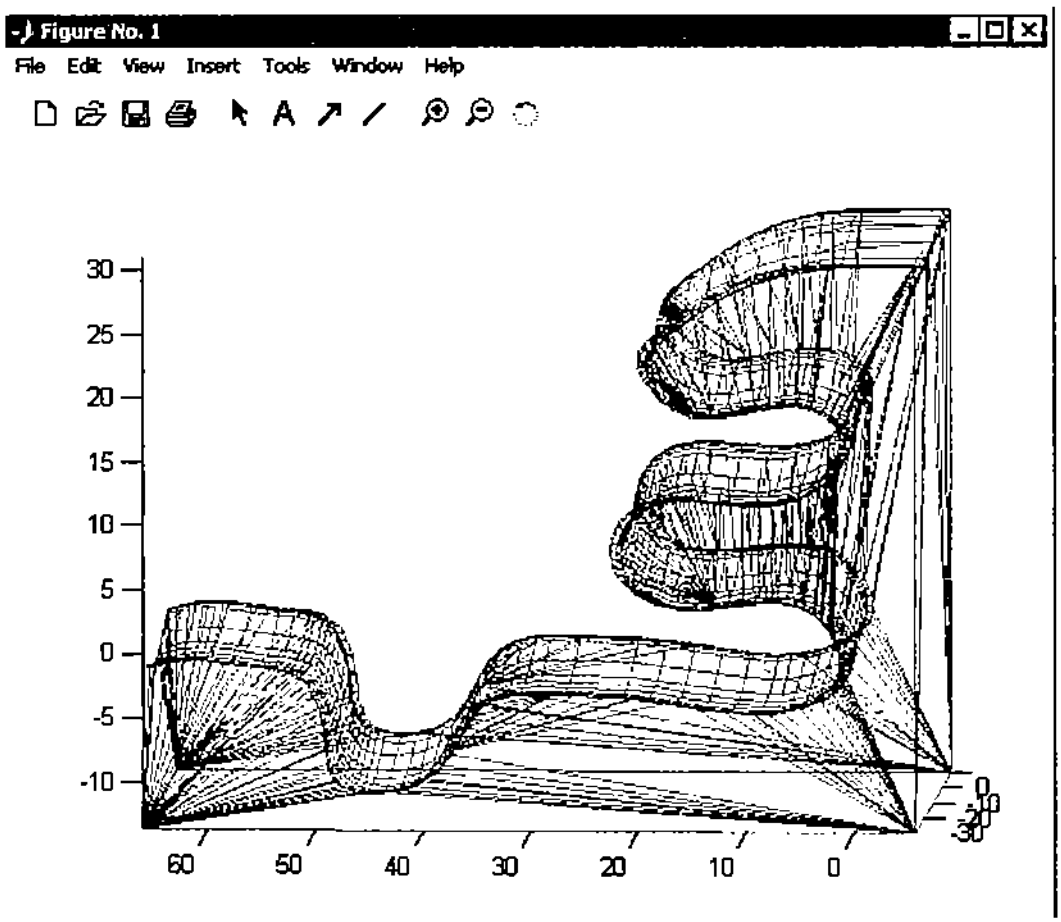


Gambar IV.4 . Tampilkan Normal vector

4.2.3. Deteksi *Close Bounded Vvolume*

Deteksi Close *Bounded volume* (daerah batas tertutup) ditandai dengan adanya segitiga bertumpuk dan solidifikasi lebih dari satu. Ilustrasi alur program adalah sebagai berikut :





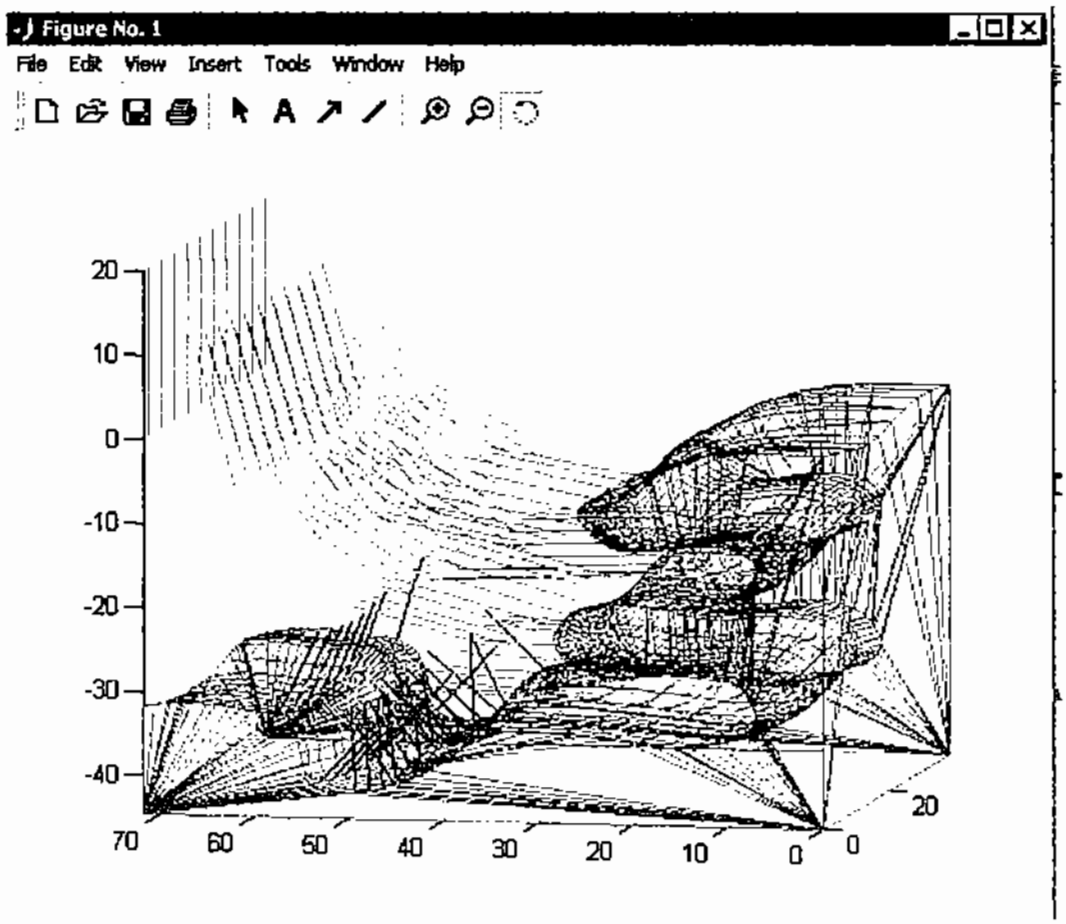
Gambar IV.5. Hasil Plot dari pendeteksian terhadap Close Bounded volume

4.2.4. Implementasi Vektor Pahat

Implementasi *vector* pahat didapat berdasarkan nilai koordinat awal pahat yang digunakan dan dihubungkan kedalam koordinat deteksi, seperti pada hasil dibawah ini :

A = x , y , z dimana A merupakan garis *vector* pahat pada posisi awal.

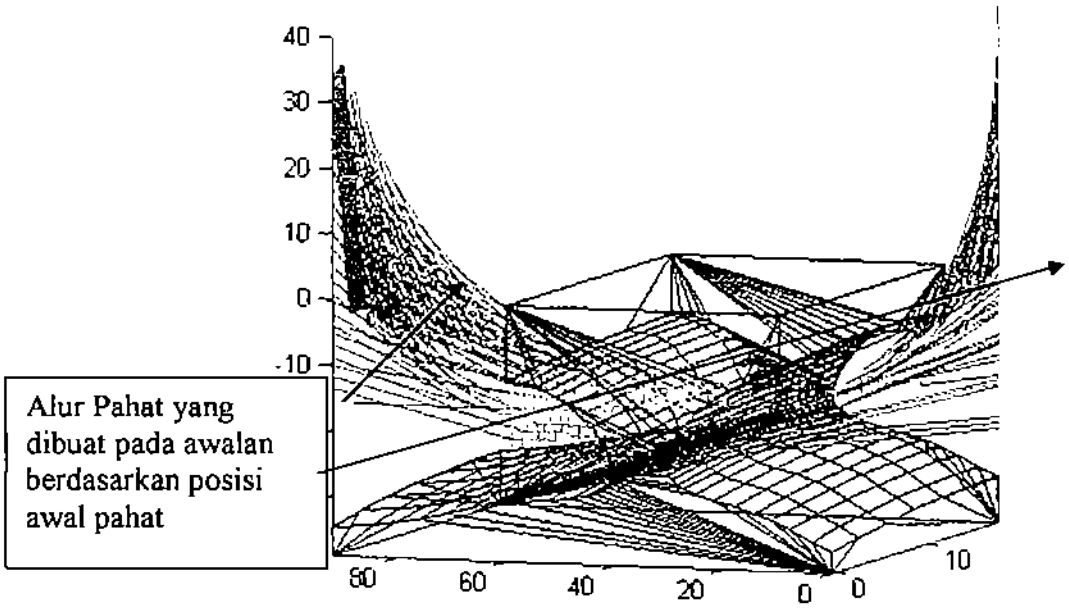
$A^1 = x^1, y^1, z^1$ dimana A^1 merupakan garis *vector* pahat pada *bounded volume*



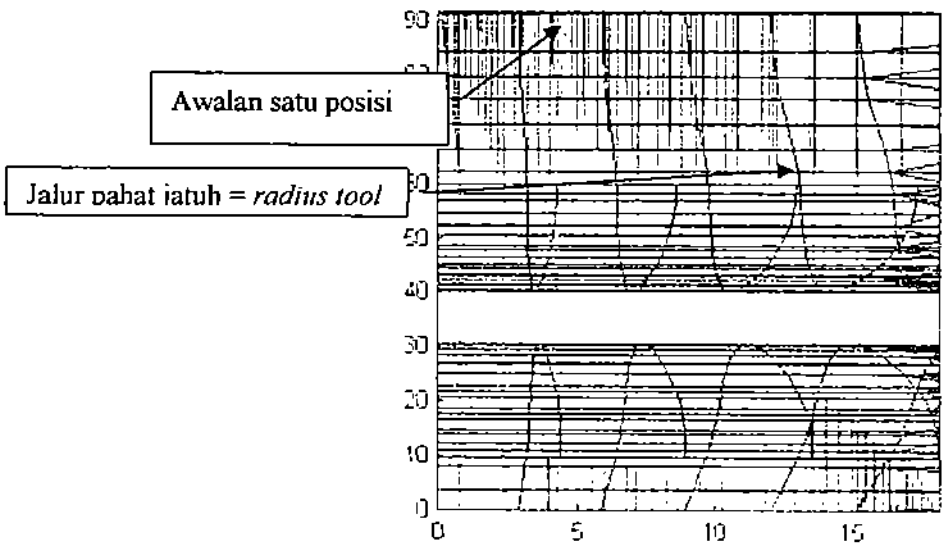
Gambar. IV.6. a Menunjukkan gerak orienasi vector pahat.

Informasi vektor pahat yang didapat diproyeksikan kembali pada awalan 1 posisi, sehingga proses jalannya pahat bisa dilihat pada awalah satu titik seperti pada gambar dibawah ini (gambar IV.6), pada bagian tersebut pembacaan segitiga dari ilustrasi untuk pendeteksian adanya bagian sold yang tidak dikerjakan bagian tengah adalah dengan metode yang sama yang digunakan pada bidang proeksi

X,Y, hanya saja Proyeksi tersebut diterapkan pada bidang Y,Z atau X,Z (terhadap sisi samping, sehingga bisa membaca koordinat jatuhnya *tool*/pahat, kemudian ditarik vektor ke koordinat posisi awal *tool*



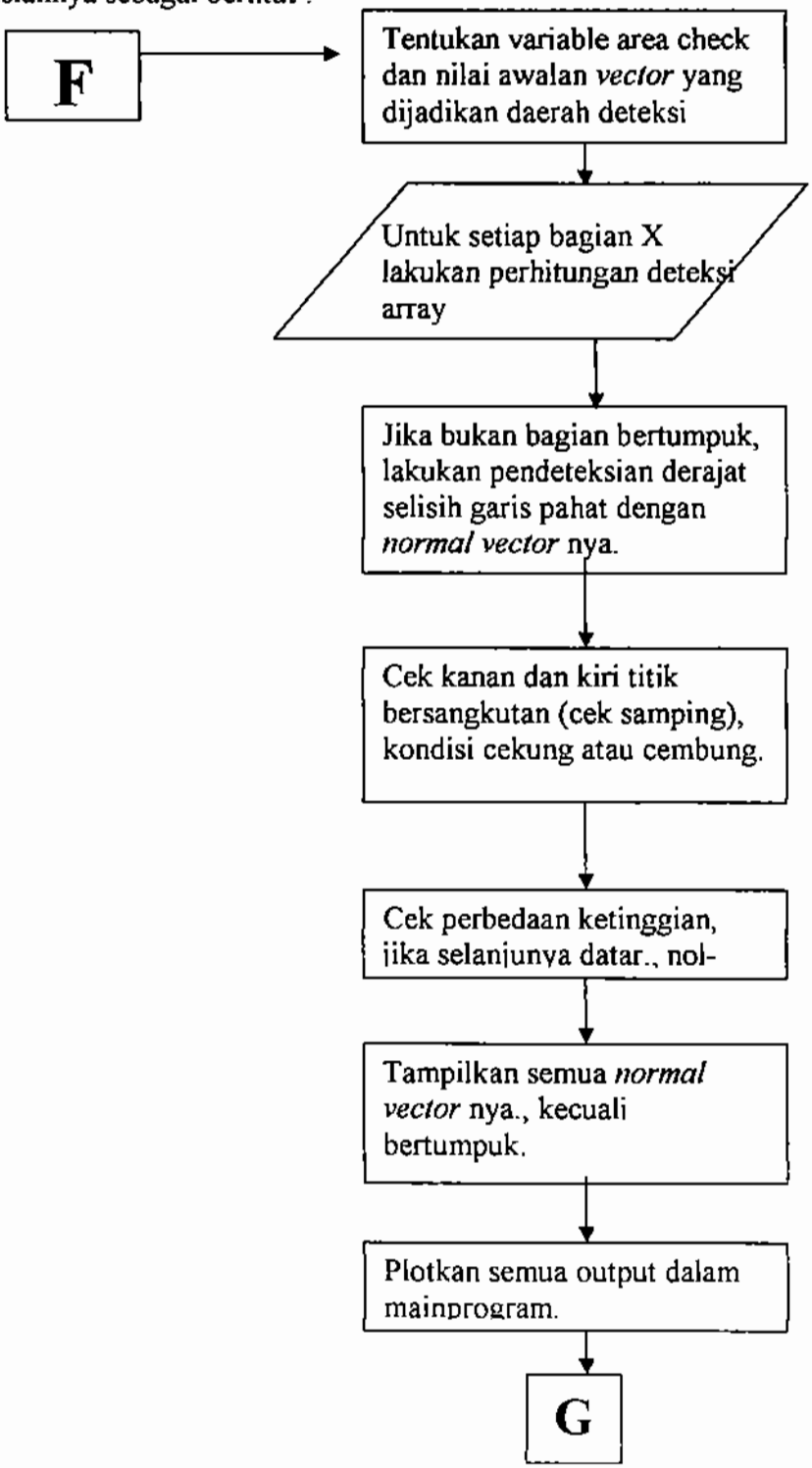
Gambar IV.6b. Gambar Alur Pahat dibuat dari posisi awal Pahatdari 2 arah bebas(ditengah terdapat solid)

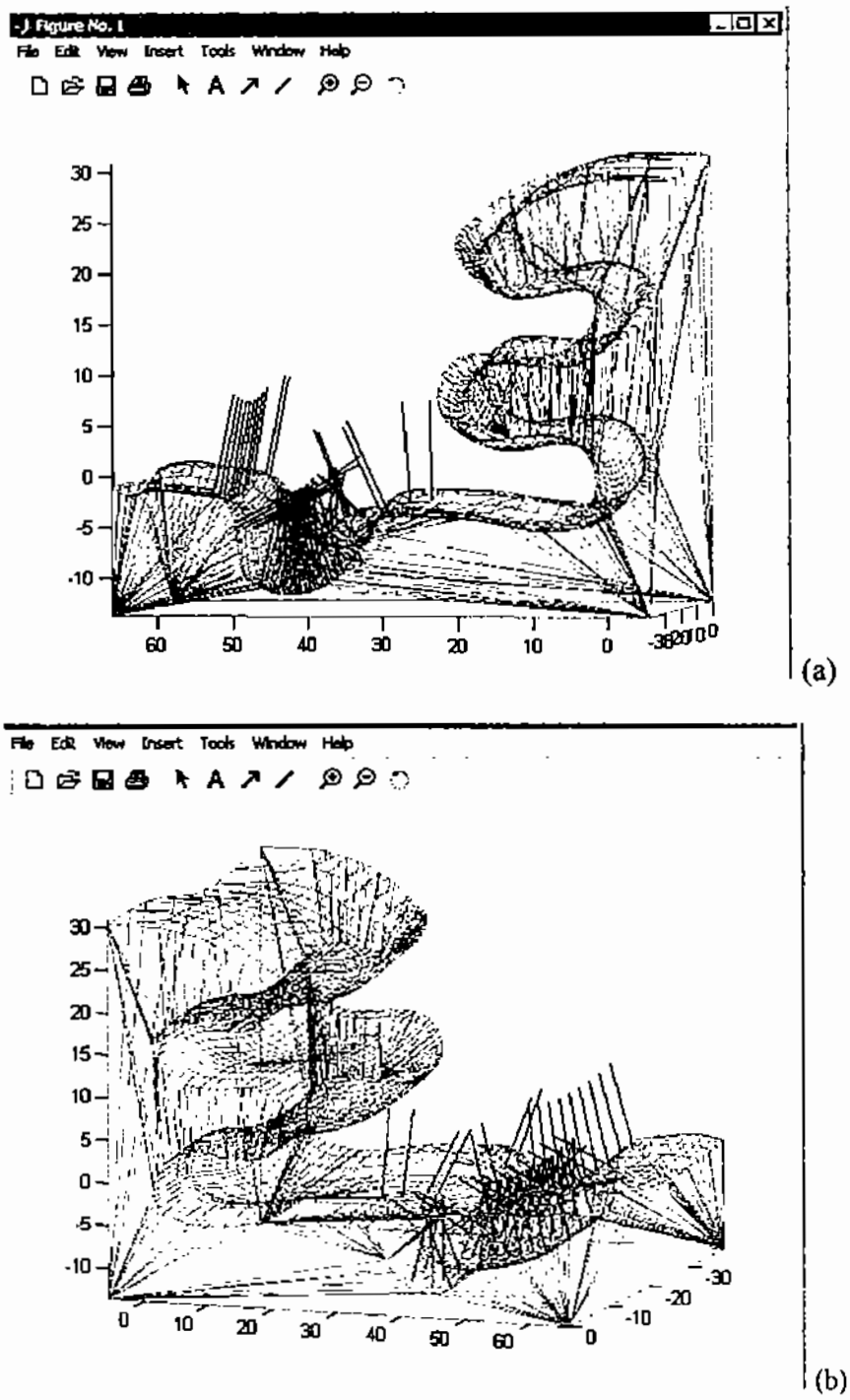


Gambar IV.6c. Pandangan Dari atas, posisi jatuh pahat antar pahat berjarak sesuai dengan lebar radius tool

4.2.5. Deteksi *Open Bounded Vvolume*

Langkah selanjutnya melakukan pengecekan untuk bagian *surface* yang tidak bertumpuk tetapi memiliki *normal vector* yang berbeda-beda arah (tidak datar) dimana hal ini diidentifikasi sebagai kesadelan (*Open Bounded volume*), langkah pendeteksiannya sebagai berikut :

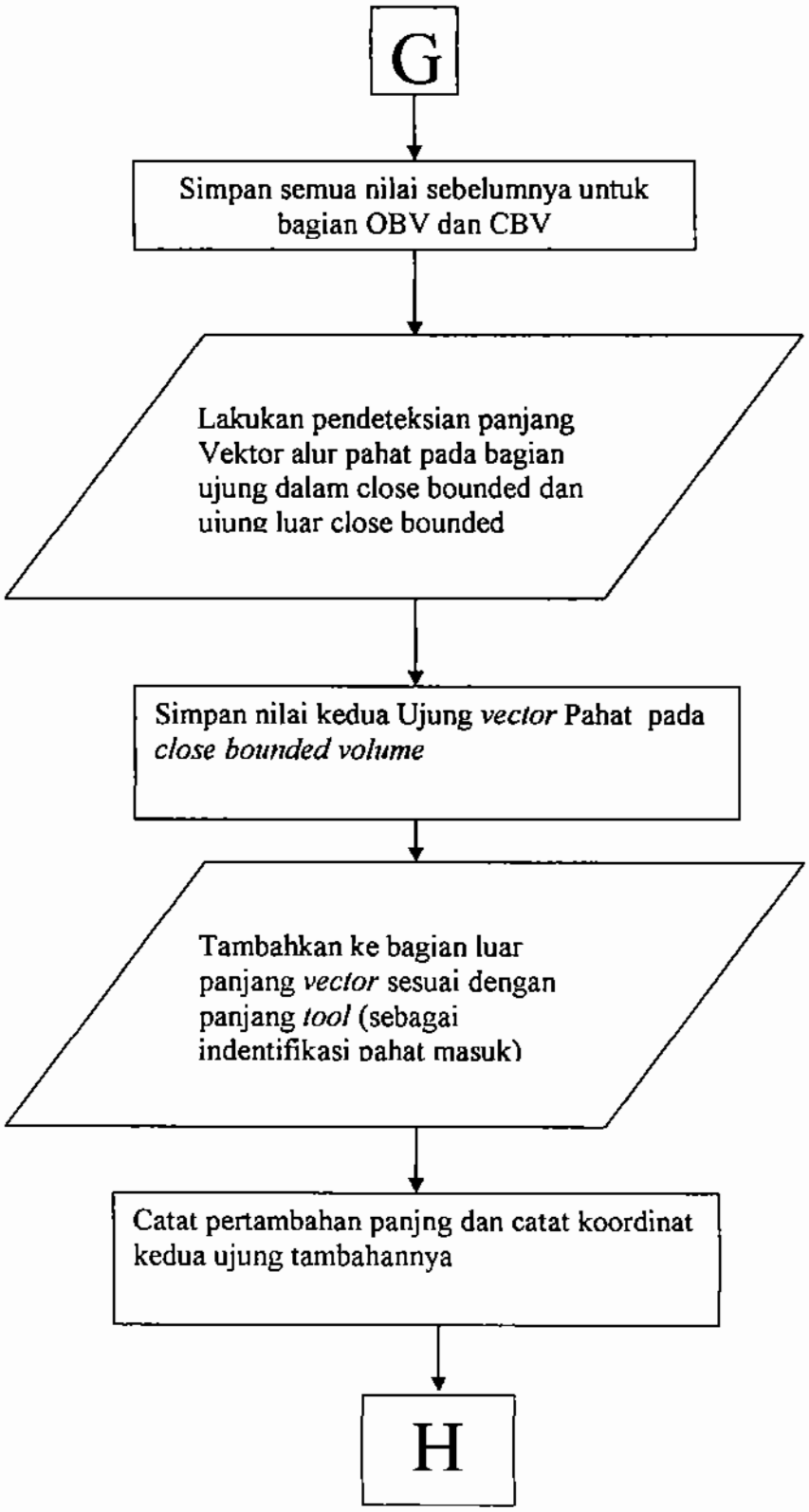


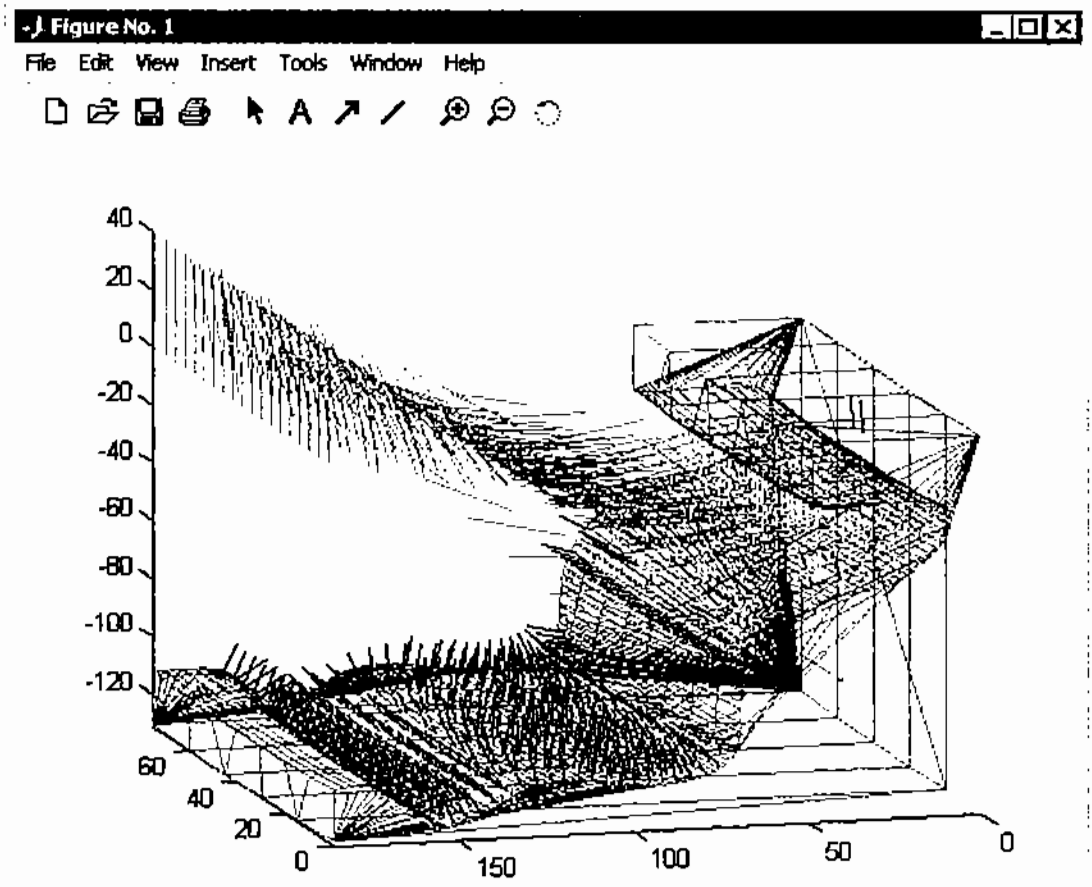


Gambar IV.7 Tampilan Deteksi Open Bounded volume (a) tampak samping kanan dan (b) ampak samping kiri

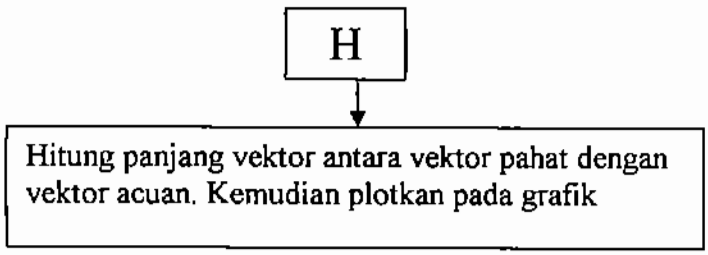
Pendeteksian ini juga diimplementasikan pada bisang sadel point dengan ketentuan teri yang mengacu pada sub-bab III.6.1 dan dengan sarat apakah arah *normal vector* berhadapan atau saling membelakangi sebesar sudut yang dipersyaratkan.

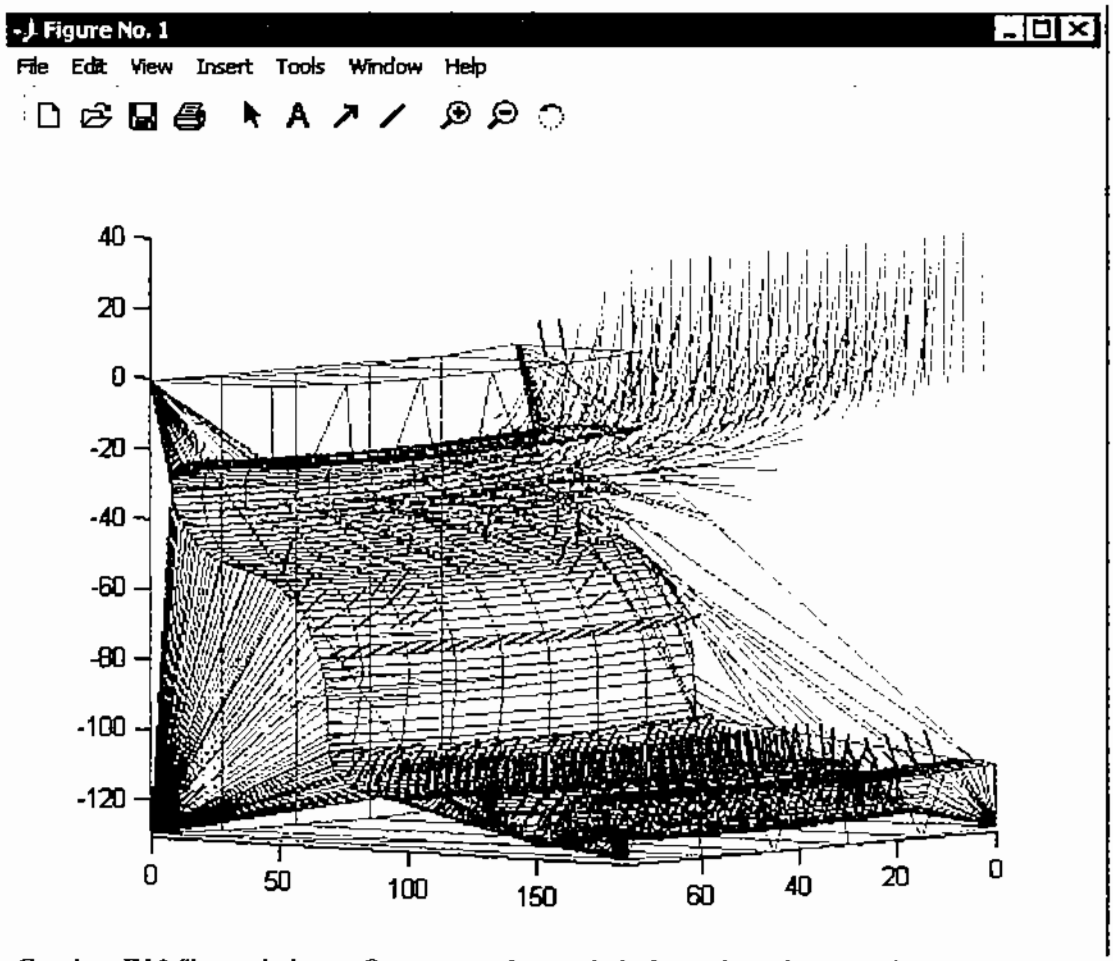
Perencanaan pendeteksian orientasi pahat yang diperlihatkan pada gambar IV.6, dimana alur *vector* pahat diplotkan dan dibuat berorientasi sejak posisi pahat awal sampai masuk pada bidang *close bounded*.





Gambar IV.8 Tampak samping Orientasi pahat pada bidang close dan open bounded





Gambar IV.9 Tampak depan Orientasi pahat pada bidang close dan open bounded satu tingkat

BAB V
SIMULASI *GRAPHIC USER INTERFACE* Dan *STAND ALONE APPLICATION*

V.1 Graphic User Interface

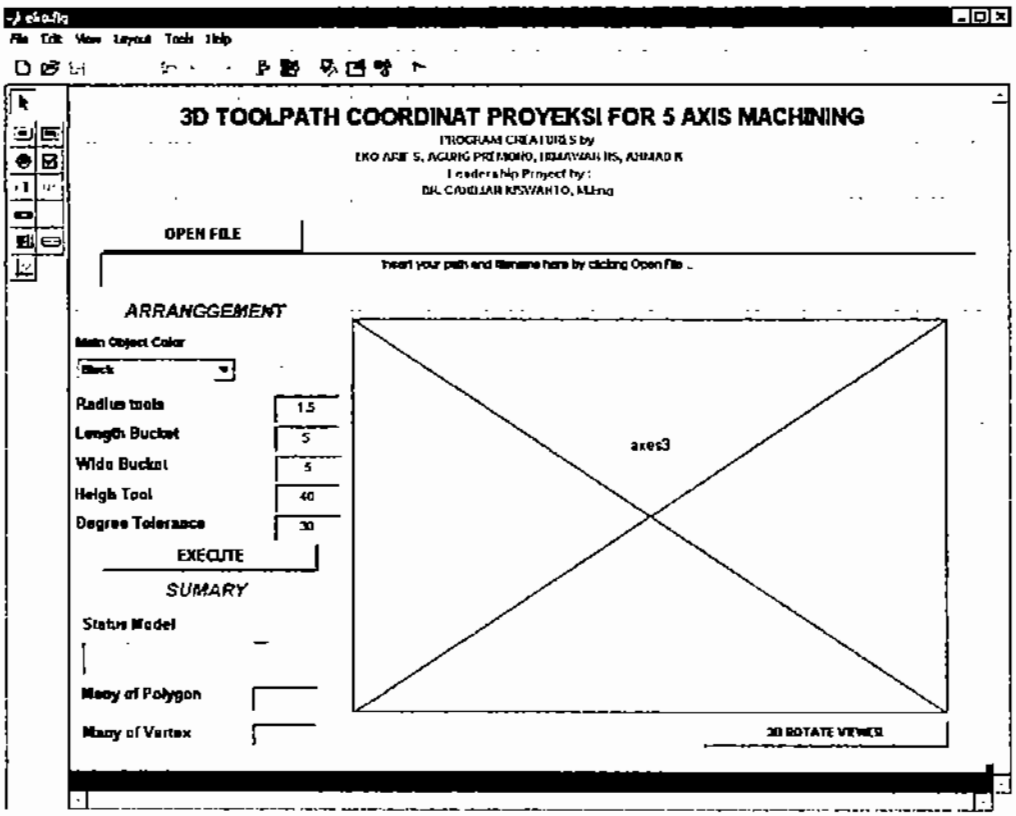
Dalam pembuatan sebuah aplikasi untuk kebutuhan dasar sangat diperlukan tampilan grafis yang sangat menunjang untuk memudahkan pengoperasian suatu system program dan sebagai media komunikasi antara program yang dibuat dengan user penggunanya. Seperti halnya aplikasi berbasis *programming* (bahasa pemograman) lainnya, Software *Matrik Laboratory* diantaranya MATLAB (dalam *platform Windows*) atau SKYLAB (Dalam *Platform linux*) juga mempunyai *feature* dasar dukung pembuatan aplikasi berbasis GUI (*Graphical User interface*), dimana dalam GUI tersebut memuat sejumlah informasi yang meminta user untuk memasukkan inputan dan sejumlah informasi lainnya yang menginformasikan kepada user sebagai keluaran program. Sama halnya dengan program pembuat aplikasi lainnya, MATLAB mempunyai kemudahan dalam pembuatan *Grafic User Interface (GUI)* ini dan hasil dari pembuatan GUI ini menghasil 2 file yaitu : berekstensi .fig sebagai figure (tampilan) dan extensi .m sebagai *callback* fungsinya (fungsi panggil) yang keduanya memiliki satu nama. Dalam pembuatan GUI, programming harus menentukan langkah-langkah diantaranya :

a. Desain tampilan (*layout*) GUI yang diinginkan
b. Komunikasi Perangkat sentuh (*Touch button*) dan lainnya (*other function button*).
c. Urutan Program utama (*Main Programming)* beserta sub programnya yang diletakkan didalam fungsi panggil (*callback*)
d. *Feature* output dalam bentuk output file atau grafik koordinat.
e. Informasi output lainnya yang diperlukan programming atau user lainnya
f. *Platform* dasar system yang digunakan (system operasi yang mendukung dan aplikasi multi support lainnya yang dibutuhkan). Kategori ini tidak selalu sama dengan bahasa pemograman lainnya, karena MATLAB perlu dukungan compiler lain untuk membuat dirinya bisa menjalankan *STAND-*

ALONE Application. Seperti halnya compiler yang digunakan yaitu *Free Command Tool (from Borland International Language Programming)*

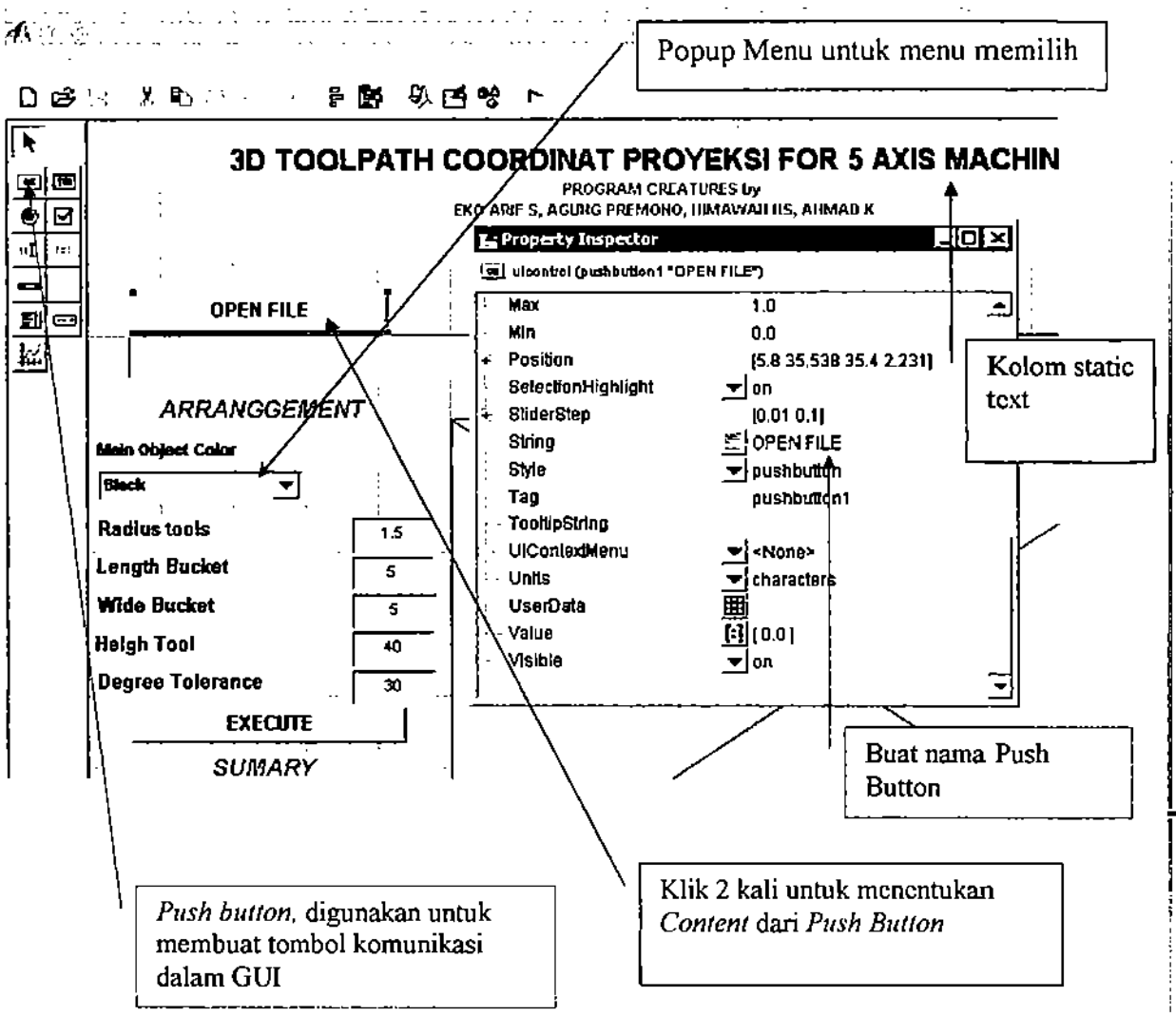
5.1.1 Desain Layout

Penentuan desain layout diusahakan sekomunikatif mungkin, dimana semua yang terkandung dalam layout tersebut adalah bentuk yang mudah dimengerti user, seperti contoh dibawah ini :



Gambar V.1 layout sederhanna yang digunakan penulis untuk berkomunikatif antara operator dengan program inti.

Langkah yag dibuat diantarnya ;



Gambar V.2 Menunjukkan fungsi callback

Langkah pertama buat semua design seperti contoh diatas :

1. Gunakan *feature* : *pushbutton, static text, edit text, popup menu, grafich.*
2. Edit semua *feature* tersebut dengan menggunakan (*double klik*) atau klik 2 kali pada bagian masing-masing item untuk menentukan identitas item tersebut.
3. Buat item tersebut seperti ilustrasi table dibawah ini :

Dalam table dibawah ini adalah menentukan Informasi **ID Tag** dalam operator fungsi GUI yang digunakan, dimana *Tag* tersebut adalah sebuah Nama sebagai Pengenal dan penghubung antara Program inti dan GUI.

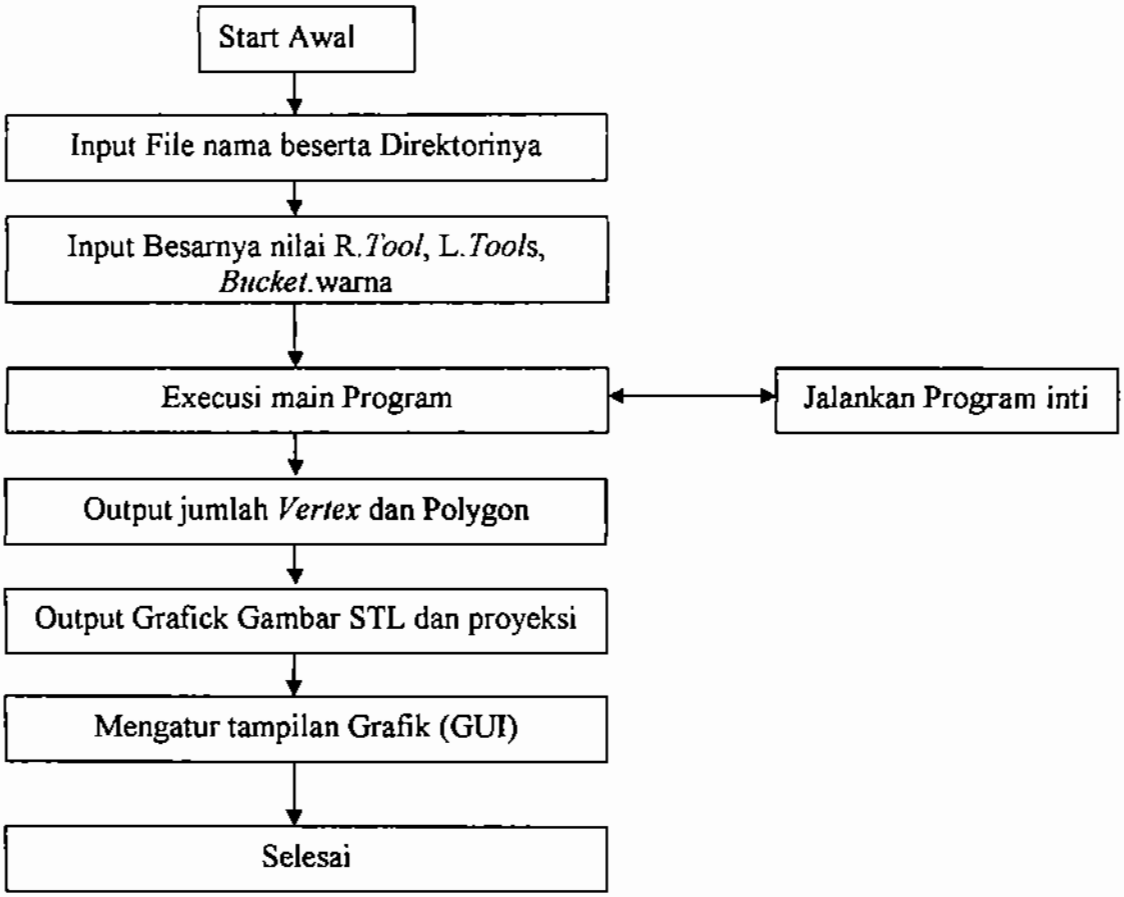
Tabel V.1 Operator Tag dan parameternya

SEBAGAI *INPUT* OPERATOR			
Item	Tentukan :	Contoh tentukan:	Keterangan :
Edit *Text* (1) / open	Tag	iFile	Edit menu untuk memasukkan nama file
Edit *Text* (2) / *Radius Tools*	Tag	R*Tool*	Edit *text* untuk memasukkan nilai R.*tools*
Edit *Text* (3) / Length *Tools*	Tag	L*Tool*	Edit *text* untuk memasukkan nilai panjang .*tools*
Edit *Text* (4) / Wide *Bucket*	Tag	W*Bucket*	Edit *text* untuk memasukkan nilai lebar *bucket*
Edit *Text* (5) / Length *Bucket*	Tag	L*Bucket*	Edit *text* untuk memasukkan nilai panjang *bucket*
SEBAGAI *OUTPUT* OPERATOR			
Item	Tentukan :	Contoh tentukan:	Keterangan :
Edit *Text* (6) / Status Model	Tag	StatusModel	Edit *text* untuk output status model (Apakah Solid atau *Surface*)

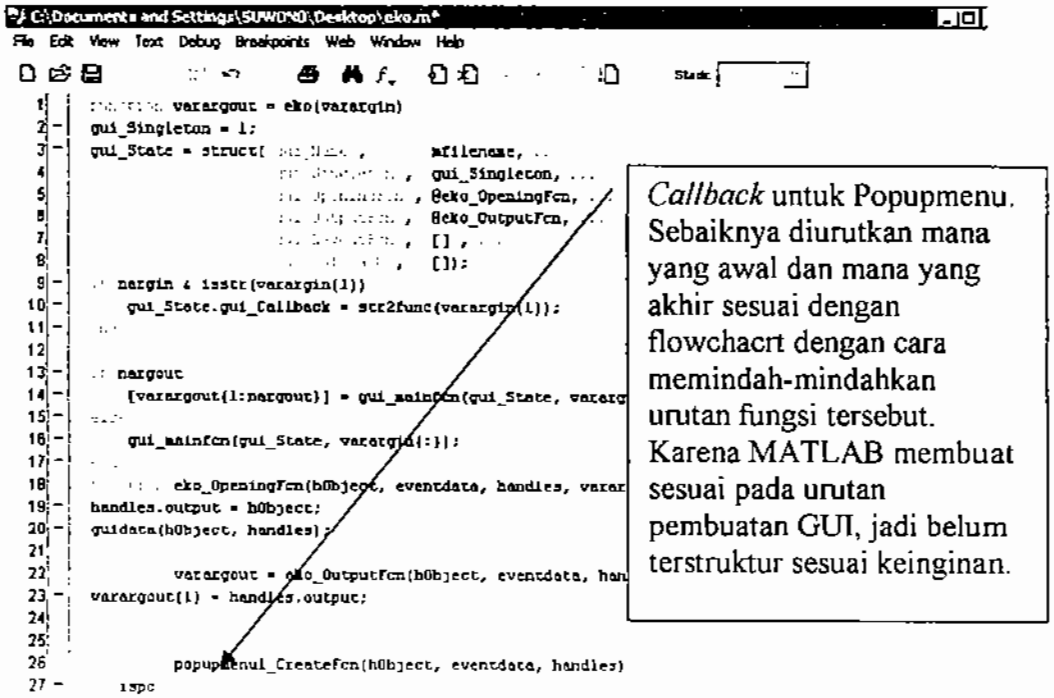
Edit *Text* (7) / Banyaknya Polygon (segitiga) di obyek	Tag	Polygon	Edit *text* untuk output Informasi banyaknya segitiga dalam output dari obyek model
Edit *Text* (8) / Banyaknya *Vertex* di obyek	Tag	*Vertex*	Edit *text* untuk output Informasi banyaknya *vertex* dalam output dari obyek model

5.1.2 Komunikasi Perangkat Sentuh

Menentukan Komunkasi antara GUI dan Program Inti merupakan bagian terpenting sebagai komunikasi perangkat keseluruhan. Dalam menetukan komunikasi antara program inti dan GUI, programmer harus paham betul bagaimana pengurutan dalam sebuah fungsi panggil (*callback*) dalam GUI di Program MATLAB dengan Program Inti yang dimasukkan kedalamnya. Pertama programmer harus menentukan dulu alur mana yang dikerjakan diawal dan alur mana yang dikerjakan diakhir (biasanya sebagai Output) yang dapat digambarkan dalam sebuah *flowcharts* kerja, seperti pada gambar dibawah ini :



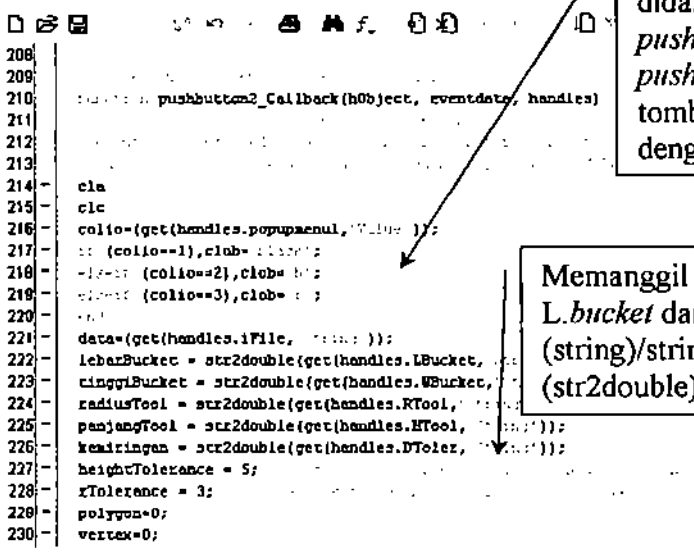
Dalam *Callback* filenya diilustrasikan sebagai berikut :



Gambar V.3. Operasi Callback dan setting

Strucktur Fungsi-sungsi dari masing-masing Button dan edit *text* yang telah dibuat dalam layout GUI akan dibuatkan secara otomatis *callback*-nya oleh MATLAB dan programming dimudahkan hanya tinggal memasukkan item atau ketentuan yang diperlukan fungsi.

Menghubungkan main program (program int



Program inti diletakkan didalam fungsi callback *push_button2*, dimana *pushbutton 2* merupakan tombol yang diberi nama dengan **execute**

Memanggil fungsi edit text dengan Tag: L.*bucket* dan dimuat dalam bentuk (string)/string dengan ketelitian double (str2double)/(0,000.....)

Gambar V.4a. MEnunjukkan command perangkat sentuh (Input GUI)

Set Output dalam GUI :

```
    count=1;
elseif strcmp(a,
    i=i-1;
    j=j-1;
elseif strcmp(a,
    T(i,1)=arr(1);
    T(i,2)=arr(2);
    T(i,3)=arr(3);
    i=i+1;
    plot3(X,Y,Z,clob);
    hold on;
    polygon=polygon+1;
elseif strcmp(a,
    ii=ii+1;
```

Dalam program inti menentukan variable sebagai output, disini penulis memberi nama polygon sebagai jumlah polygon, dimana bila bertemu kalimat 'endfacet' dalam STL maka nilai polygon sekarang nilainya = polygon sebelumnya +1
Dan *loop* ini akan berulang sampai pembacaan STL selesai.

Penempatan informasi di GUI layout :

```
            A(1) = garisMembujur(i,1,1);
            B(1) = garisMembujur(i,1,2);
            C(1) = garisMembujur(i,1,3);

            A(2) = garisMembujur(i,1,1)+vektorGaris(1)*50;
            B(2) = garisMembujur(i,1,2)+vektorGaris(2)*50;
            C(2) = garisMembujur(i,1,3)+vektorGaris(3)*50;

            plot3(A,B,C,'w');
            hold on;
        end
    end

end
set(handles.polygon,'String',polygon);
set(handles.vertex,'String',vertex);
axis([xMinMax(1)+deltaX xMinMax(2)+deltaX yMinMax(1)+deltaY yMinMax(2)+deltaY zMinMax(1)+deltaZ z
```

Gambar V.4b Memunjukkan command perangkat sentuh (Output GUI)

Set Nilai polygon tadi dalam GUI Edit Text dengan Tag: polygon (*Case Sensitif*), perintah set nilai ini bisa dimasukkan diakhir setelah program utama sebelum masuk *Sub Function* lainnya atau paling akhir pada bagian *Callback* fungsi dari *push_button2* ini.

Begitu juga berlaku untuk *Vertex* dan lainnya yang ingin diinformasikan.

Aturan yang dipakai dalam pengambilan atau penempatan sebuah nilai atau karakter pembuatan aplikasi GUI :

Misalnya Tag : iFile (dalam edit *text*)

Untuk mengambil nilai ini maka variable yang ditentukan sebagai input nilai ini harus mengambil dari Edit menu ini :

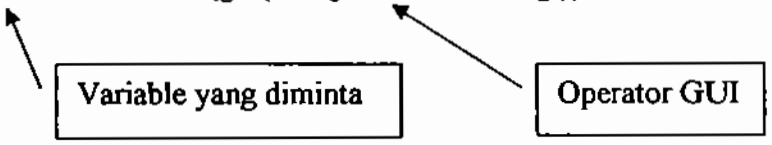
Misalnya variable : Eko

Maka bila Eko adalah nilai dari inputan tersebut commandnya adalah :

Eko=get(handles.iFile,'String') atau

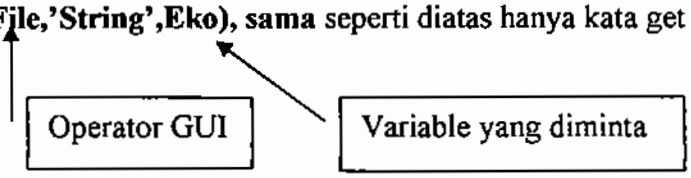
Eko=get(hObject.iFile,'String') atau

Jika iFile adalah *text* maka : **Eko=str2double(get(hObject.iFile,'String'))**



Dan bila ingin mengeset kebalikannya / nilai dari variable Eko ingin ditampilkan pada edit_*text* dengan Tag:iFile, maka :

set(handles.iFile,'String',Eko), sama seperti diatas hanya kata get diganti set.



command perintah tersebut diletakkan pada saat nilai yang diminta untuk ditampilkan diexecusi dalam program, atau kalimat (set(handles.....)) tersebut ada dalam program intinya.

Setelah semua aplikasi selesai dibuat kemudian dicoba untuk mengetahui hasilnya apakah benar sesuai dengan yang diminta dan telah sesuai prosedur yag diinginkan. Bila ternyata terdapat beberapa kesalahan program *Matrik Laboratory* ini akan menampilakannya pada *command window work* dimana letak kesalahan tersebut, dan programming dapat memperbaikinya dengan segera.

1.2 *STAND-ALONE* Application

Adalah mengkonversikan code program yang kita buat kedalam sebuah file yang dapat ber-exekusi sendiri dimana file tersebut berupakan file berextension exe, sehingga dapat berjalan walaupun tanpa menggunakan program *flatform*-nya (program utama dukungannya/pengendalinya)

Sehingga dinamakan Aplikasi yang berdiri sendiri (*STAND ALONE APLICATION).* Kode pemograman yang dikonverrsi dari MATLAB ke dalam C atau C++ dibuat otomatis menggunakan *Software* MATLAB. Hasil dari Pembuatan *Stand-Alone Application* ditunjukkan pada gambar V.5a,5b,6a dan 6b

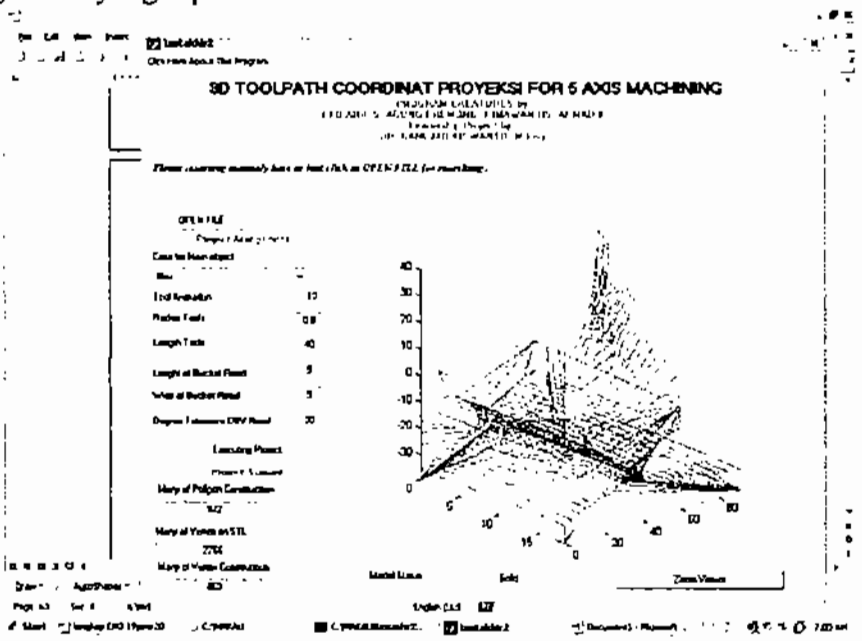
Contoh Command :

Mcc –m –B sglcpp Filename.m

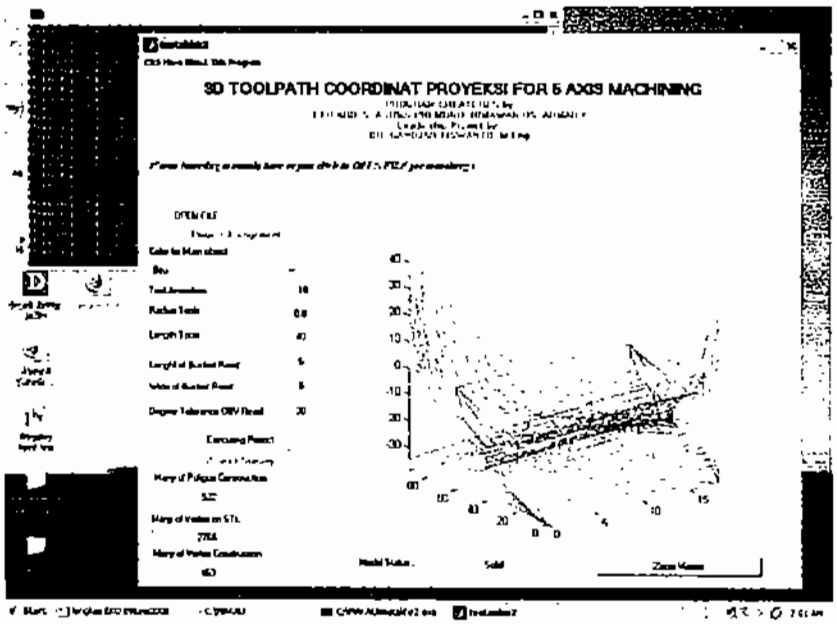
Hasil keluarannya :

Filename.exe

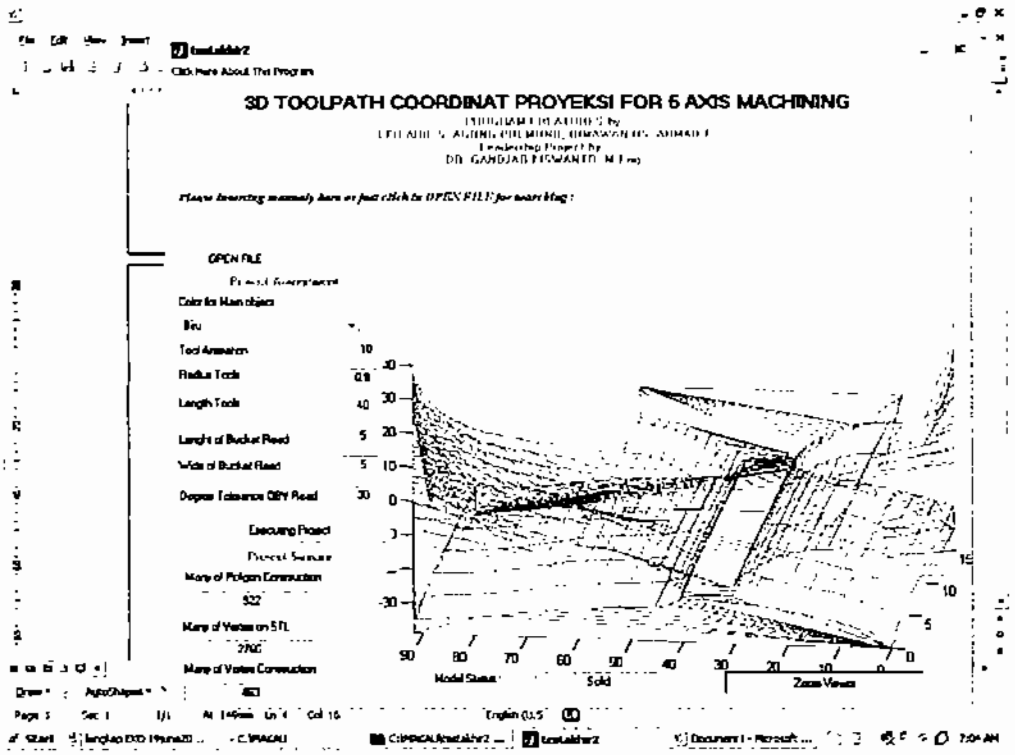
Sehingga hasil yang diperoleh :



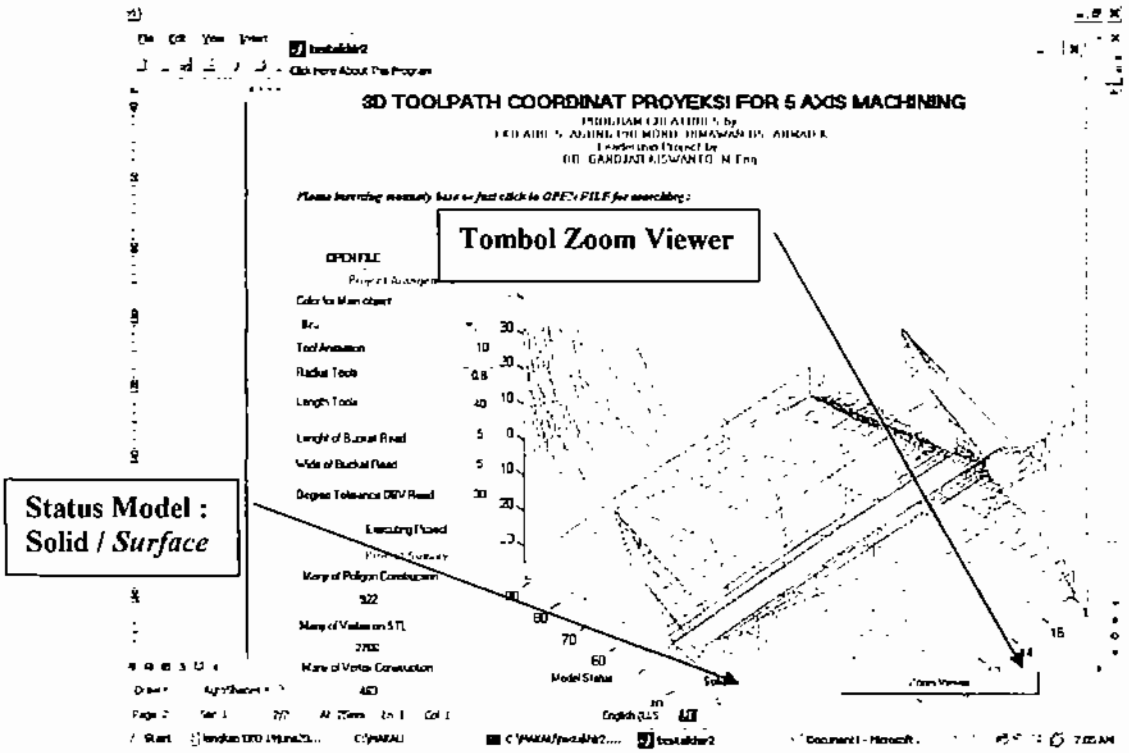
Gambar V.5a Contoh aplikasi Stand Alone yang bergerak dalam Platform WindowsXP (TestAkhir.exe)



Gambar V.5b Contoh aplikasi Stand Alone yang bergerak dalam Platform WindowsXP (TestAkhir.exe)



Gambar V.6a Mode ZoomViewer diaktifkan.



Gambar V.6b Mode ZoomViewer diaktifkan tampak atas.