Makalah Tugas Akhir

PERANCANGAN PERMAINAN CANNON 3D PADA SISTEM OPERASI LINUX MENGGUNAKAN API OPENGL DAN LAZARUS 0.9.22

Aghus Sofwan[1], Agung Budi P.[1], Fitriadi Nugroho[2]

***Abstrak* – Saat ini permainan 3D lah yang mendominasi aplikasi permainan komersial di dunia. Melihat keadaan seperti ini, Penulis kemudian mengembangkan sebuah permainan Canon 3D. Untuk membuat sebuah permainan berbasis 3D di Linux dibutuhkan API (Application Programming Interface) yang dapat menjembatani antara aplikasi dan kartu grafis. Saat ini API grafis 3D yang populer adalah DirectX yang hanya bisa digunakan di Windows dan OpenGL yang bersifat multi platform. Karena Sifatnya yang multi platform maka untuk mengembangkan permainan 3D di Linux digunakan API OpenGL.**

Tanpa sistem deteksi tubrukan yang baik, permainan berbasis 3D akan menjadi kurang sempurna dan akan membuat permainan menjadi kurang realistis. Oleh karena itu perlu dirancang sebuah sistem deteksi tubrukan yang dapat menangani tubrukan dengan baik tanpa terlalu menghambat kinerja sistem. Sistem deteksi tubrukan ini dirancang dengan menggunakan bounding volume dengan tipe Axis Aligned Bounding Box (AABB) dan metode tes perpotongan segitiga menggunakan metode interval overlap yang dikembangkan oleh Tomas Moller.

Berdasarkan dari hasil pengujian yang dilakukan, diperoleh kesimpulan bahwa telah dapat dirancang dan dibangun sebuah aplikasi permainan Cannon 3D dengan sistem deteksi tubrukan pada sistem operasi Linux dengan menggunakan API grafis OpenGL dengan menggunakan Lazarus. Sistem deteksi tubrukan yang dibangun berhasil mendeteksi tubrukan yang terjadi dengan baik. Pada saat mendeteksi tubrukan walaupun sistem berjalan lambat tetap dapat mendeteksi tubrukan dengan baik.

Kata kunci: Permainan, OpenGL, Lazarus, Linux, metode interval overlap

I. PENDAHULUAN

1.1 Latar Belakang

Seiring dengan kemajuan dalam teknologi 3D, perkembangan dalam dunia pemrograman permainan juga berubah dengan cepat. Deteksi tubrukan merupakan salah satu dasar dari permainan 3D. Deteksi tubrukan akan memberikan nuansa yang lebih realistis dalam permainan 3D, misalnya objek tidak dapat memotong objek lain, atau objek tidak akan melayang pada saat seharusnya jatuh. Penerapan deteksi tubrukan yang buruk dapat menyebabkan permainan menjadi kurang realistis dan menjadi kurang menarik.

Metode deteksi tubrukan sangat banyak jumlahnya, pemilihan metode deteksi tubrukan didasarkan pada bentuk primitif yang digunakan dan tingkat ketelitian yang diinginkan. Primitif yang digunakan dalam Permainan Cannon 3D adalah segitiga, oleh karena itu digunakan metode untuk menghitung perpotongan antara dua segitiga. Metode untuk menghitung perpotongan antara segitiga ada beberapa macam, salah satunya adalah metode *separating axis theorem* dan metode *interval overlap* yang dikembangkan oleh moller. Dalam aplikasi ini digunakan akan dibuat deteksi tubrukan dengan metode *interval overlap*. Dengan metode ini akan dibuat suatu sistem deteksi tubrukan dan akan diuji kecepatan metode ini untuk menangani deteksi tubrukan pada permainan.

Pembuatan sebuah permainan berbasis 3D di sistem operasi linux membutuhkan API (*Application programming Interface*) yang dapat menjembatani antara aplikasi dan kartu grafis. Saat ini API grafis 3D yang popular adalah DirectX yang hanya bisa digunakan pada sistem operasi Microsoft windows dan OpenGL yang bersifat *multi platform*. Karena sifatnya yang *multi platform* maka untuk mengembangkan permainan di sistem operasi linux digunakan API OpenGL.

1.2 Tujuan

Tujuan dalam tugas akhir ini adalah membuat aplikasi permainan Cannon 3D dengan sistem deteksi tubrukan menggunakan metode *interval overlap.*

1.3 Pembatasan Masalah

Agar tidak menyimpang dari pokok pembahasan, pada Tugas Akhir ini Penulis membuat batasan masalah pada hal-hal sebagai berikut.

1. API grafis yang digunakan adalah OpenGL.
2. Permainan Cannon3D dibangun menggunakan bahasa pemrograman Lazarus 0.9.22
3. Antarmuka perangkat lunak permainan Canon 3D akan dirancang menggunakan GLUT (OpenGL Utility Toolkit).
4. *Bounding Volume* yang digunakan adalah *Axis Aligned Bounding Box.*

[1]Dosen Teknik Elektro UNDIP
[2]Mahasiswa Teknik Elektro UNDIP

5. Metode tes perpotongan segitiga yang digunakan adalah metode *interval overlap* oleh Tomas Moller.
6. Perangkat lunak permainan Canon 3D ini dirancang untuk berjalan di atas sistem operasi Linux.

II. DASAR TEORI

2.1 OPENGL

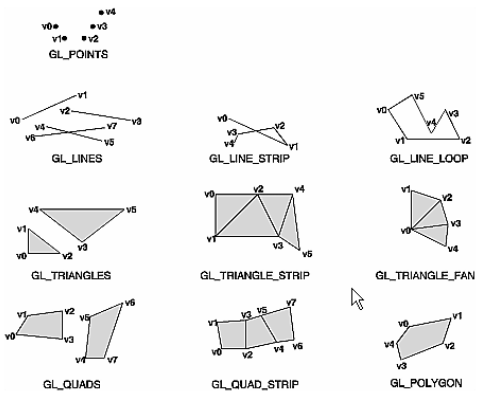
OpenGL merupakan kepanjangan dari open graphic library. OpenGL diproduksi oleh Silicon Graphics, Inc (SGI) dan pada awalnya ditujukan hanya untuk sistem komputer mereka, tetapi dalam perkembangannya, OpenGL diterima menjadi salah satu bakuan (*standard*) dalam grafika computer dan saat ini telah diimplementasikan dalam berbagai sistem computer. OpenGL merupakan pustaka program (program *library*) yang menyediakan sejumlah perintah yang berhubungan dengan grafika.

Perintah glBegin (mode) mengawali perintah menggambar. Mode merupakan konstanta yang menyatakan bagaimana OpenGL harus menghubungkan titik/*vertex* yang akan digambar. Mode yang dapat digunakan diperlihatkan pada tabel dibawah ini.

Tabel 2.1 Mode pada GLBegin

Nilai	Arti
GL_POINTS	Setiap titik diperlakukan sebagai titik terpisah
GL_LINES	Dua pasang diperlakukan sebagai garis
GL_LINE_STRIP	Sama seperti GL_LINES tetapi setiap garis dihubungkan
GL_LINE_LOOP	Sama sepeti GL_LINE_STRIP tetapi titik pertama dan terakhir membentuk garis pula
GL_TRIANGLES	Tiga pasang titik dianggap sebagai bidang segitiga
GL_TRIANGLE_STRIP	Bidang segitiga yang saling berhubungan
GL_TRIANGLES_FAN	Mirip GL_TRIANGLE_STRIP tetapi semua bidang menggunakan satu titik yang sama
GL_QUADS	Empat titik dianggap sebagai empat polygon-empat-sisi (quadrilaterals)
GL_QUAD_STRIP	Pasangan quadrilaterals
GL_POLYGON	titik dianggap sebagai titik sudut polygon

Gambar dibawah ini menjelaskan mode menggambar pada OpenGL dan bagaimana tata urutan titik-titik dihubungkan sehingga membentuk primitif.



Gambar 2.1 Mode gambar pada OpenGL

2.2 GLUT

OpenGL tidak menyediakan fungsi fungsi untuk manajemen antarmuka dan interaksi dengan pengguna. Hal ini dikarenakan setiap sistem operasi memiliki fungsi tersendiri untuk menangani OpenGL, misalnya GLX untuk Linux, Wiggle untuk Microsoft Windows, AGL, NSOpenGl, CGL untuk MacOS X dll. Karena fungsi fungsi ini spesifik untuk sistem operasi tertentu maka membuat program yang ditulis menjadi tidak multi platform. Untuk menghindari hal ini maka dapat digunakan GLUT (OpenGL Utility Toolkit) untuk membuat antarmuka yang independen.

2.3 Collision Detection

Segitiga adalah primitif yang paling banyak digunakan dalam pembuatan objek 3 dimensi, hal ini disebabkan karena sifat segitiga yang selalu konveks sehingga dapat lebih cepat diproses oleh kartu grafis. Karena sebagian besar objek dibentuk oleh segitiga maka untuk deteksi tubrukan dilakukan dengan memeriksa apakah ada segitiga segitiga dalam dua buah objek yang saling memotong, apabila ada maka pastilah terjadi tubrukan. Metode i*nterval overlap* yang dikembangkan oleh Tomas Moller merupakan salah satu metode yang digunakan untuk mendeteksi perpotongan segitiga.

Misal terdapat dua segitiga T_1 dan T_2, titik titik pada segitiga T_1 dan T_2 adalah v_0^1, v_1^1, v_2^1 dan v_0^2, v_1^2, v_2^2 , dan bidang yang ditempati adalah π_1 dan π_2.

Langkah pertama adalah menentukan persamaan bidang π_2 : $N_2.X + d_2 = 0$, dengan X adalah titik sembarang pada bidang. N_2 adalah persamaan garis normal pada bidang π_2. Untuk menghitung N_2 dan d_2 digunakan persamaan:

$$N_2 = (v_1^2 - v_0^2) \times (v_2^2 - v_0^2)$$

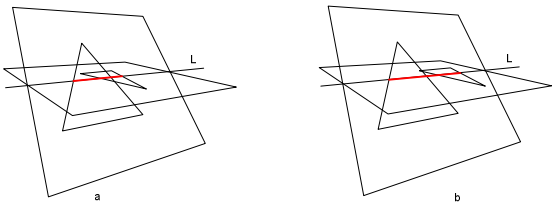
$$d_2 = -N_2.v_0^2$$

Langkah selanjutnya adalah menghitung jarak dari titik titik pada segitiga T_1 ke bidang π_2. Penghitungan jarak dilakukan dengan memasukkan titik titik T_1 pada persamaan bidang, sehingga didapat $dv_i^1, i = 0,1,2$.

$$dv_i^1 = N_2.v_i^1 + d_2,\ i = 0,1,2.$$

jika semua $dv_i^1 \neq 0, i = 0,1,2$ dan semua memiliki tanda yang sama (semuanya positif atau semuanya negatif) maka T_1 terletak pada satu sisi bidang yang sama maka tidak akan terjadi perpotongan. Hal yang sama dilakukan pula kepada segitiga T_2 dan bidang π_1. Jika semua $dv_i^1 = 0, i = 0,1,2$ maka segitiga terletak dalam bidang yang sama sehingga cukup dilakukan pemeriksaan perpotongan segitiga 2 dimensi.

Jika semua persyaratan diatas tidak dipenuhi maka pada bidang π_1 dan π_2 terdapat suatu garis perpotongan yaitu: $L = O + tD$, dimana $D = N_1 \times N_2$ adalah arah dari garis dan O adalah suatu titik pada garis itu. Karena perhitungan yang telah dilakukan di atas tadi, kedua segitiga dipastikan memotong garis L. Perpotongan tersebut membentuk *interval* pada L, jika kedua *interval* tersebut berimpit maka dapat dipastikan bahwa kedua segitiga berpotongan. Kedua kemungkinan tersebut digambarkan pada gambar dibawah ini:



Gambar 2.2 a. Kedua segitiga berpotongan.
b. Kedua segitiga tidak berpotongan.

Gambar diatas menunjukkan kedua segitiga dan bidang dimana segitiga tersebut berada, perpotongan antara kedua bidang ditunjukkan dengan garis L. Gambar a menunjukkan saat kedua segitiga berpotongan. Perpotongan antara kedua segitiga tersebut digambarkan dengan garis merah.

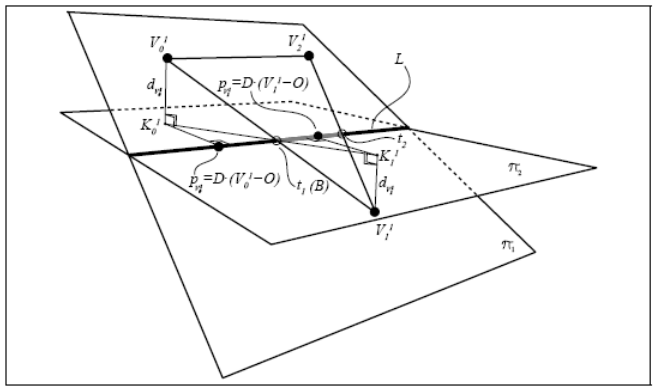
Langkah selanjutnya adalah menghitung garis perpotongan antara T_1 dan L. Misalkan v_0^1 dan v_2^1 terdapat pada sisi yang sama terhadap bidang π_2 dan v_1^1 terletak pada sisi yang lain. Untuk menemukan garis yang merupakan perpotongan dari sisi $\overline{v_0^1\, v_1^1}$ dan $\overline{v_1^1\, v_2^1}$ dan L, pertama tama semua titik diproyeksikan ke L:

$$pv_i^1 = D.\left(v_i^1 - O\right). \quad (3)$$

Langkah selanjutnya adalah menghitung nilai parameter garis, t_1, untuk $B = \overline{v_0^1\, v_1^1} \cap L = O + t_1 D$. K_i^1 menunjukkan proyeksi dari v_i^1 ke π_2, terlihat bahwa $\Delta v_0^1 B K_0^1$ dan $\Delta v_1^1 B K_1^1$ adalah sama, sehingga:

$$t_1 = pv_0^1 + (pv_1^1 (2) pv_0^1)\frac{dv_0^1}{dv_0^1 - dv_1^1}$$

Perhitungan juga dilakukan untuk segitiga T2, apa bila kedua garis saling berpotongan maka kedua segitiga juga berpotongan. Situasi diatas tampak pada gambar dibawah ini:



Gambar 2.3 Perhitungan garis perpotongan antara T1 dan L

Jika Kedua segitiga terletak pada bidang yang sama, maka kedua segitiga tersebut diproyeksikan kebidang yang terletak pada salah satu sumbu. Pemilihan sumbu diatur sehingga luas segitiga menjadi maksimal. Selanjutnya tes perpotongan segitiga 2 dimensi dilakukan. Pertama tes semua sisi dari T1 untuk perpotongan dengan T2. Jika perpotongan ditemukan maka kedua segitiga berpotongan. Jika tidak maka harus diperiksa apakah T1 berada dalam T2 atau sebaliknya.

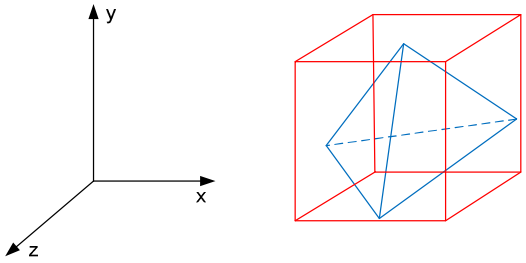
Algoritma dari metode ini adalah:

1. Hitung persamaan bidang segitiga 2
2. Hitung jarak semua titik segitiga 1 terhadap bidang segitiga 2
3. Jika jarak titik semua bertanda positif atau semua bertanda negatif, maka segitiga 1 terletak pada satu sisi bidang segitiga 2, maka segitiga tidak berpotongan.
4. Hitung persamaan bidang segitiga 1
5. Hitung jarak semua titik segitiga 2 terhadap bidang segitiga 1
6. Jika jarak titik semua bertanda positif atau semua bertanda negatif, maka segitiga 2 terletak pada satu sisi bidang segitiga 1, maka segitiga tidak berpotongan.
7. Hitung garis perpotongan antara bidang 1 dan bidang 2
8. Periksa apakah segitiga 1 dan 2 terletak pada bidang yang sama, jika ya, maka lakukan tes

perpotongan segitiga 2 dimensi, jika tidak lanjut ke langkah selanjutnya.
9. Proyeksikan garis perpotongan ke sumbu terbesar
10. Hitung *interval* untuk masing masing segitiga
11. Jika *interval* berimpit, maka segitiga berpotongan, jika tidak, segitiga tidak berpotongan.

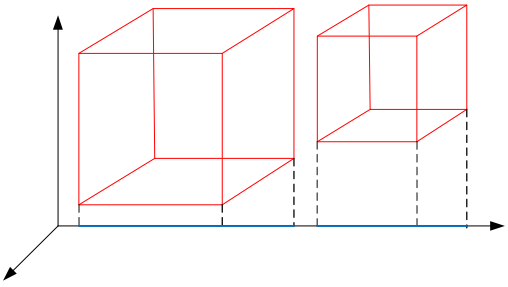
Dalam aplikasi 3D biasanya jumlah primitif yang digunakan sangat besar. Sehingga untuk melakukan perhitungan deteksi perpotongan antara seluruh segitiga yang ada dalam satu *frame* tidaklah mungkin karena akan memerlukan kecepatan perhitungan yang sangat besar. Untuk itu diperlukan suatu cara untuk mengurangi jumlah perhitungan dalam satu *frame*, dengan cara membuat suatu pembatas antara objek objek yang akan bertabrakan, sehingga deteksi perpotongan hanya dilakukan antara objek objek yang sudah berdekatan.

Axis Aligned Bounding Box (AABB) merupakan salah satu bentuk dari *Bounding Volume* AABB berbentuk kotak yang semua sisinya searah dengan sumbu, dan setiap sisinya tegak lurus dengan salah satu sumbu koordinat. Karena AABB selalu searah dengan sumbu, maka ketika objek ditransformasikan, misalnya berotasi maka AABB tidak bias dirotasikan juga, AABB harus dihitung ulang setiap *frame*. Perhitungan AABB relatif sederhana oleh karena itu hanya sedikit berdampak pada kecepatan proses keseluruhan.



Gambar 2.4 Sebuah objek 3D (biru) dan AABB nya (merah).

Untuk mengetahui apakah 2 AABB saling bertubrukan maka digunakan langkah perhitungan seperti di bawah ini:
1. Proyeksikan semua sisi ke sumbu x.



Gambar 2.5. Proyeksi AABB ke sumbu x,

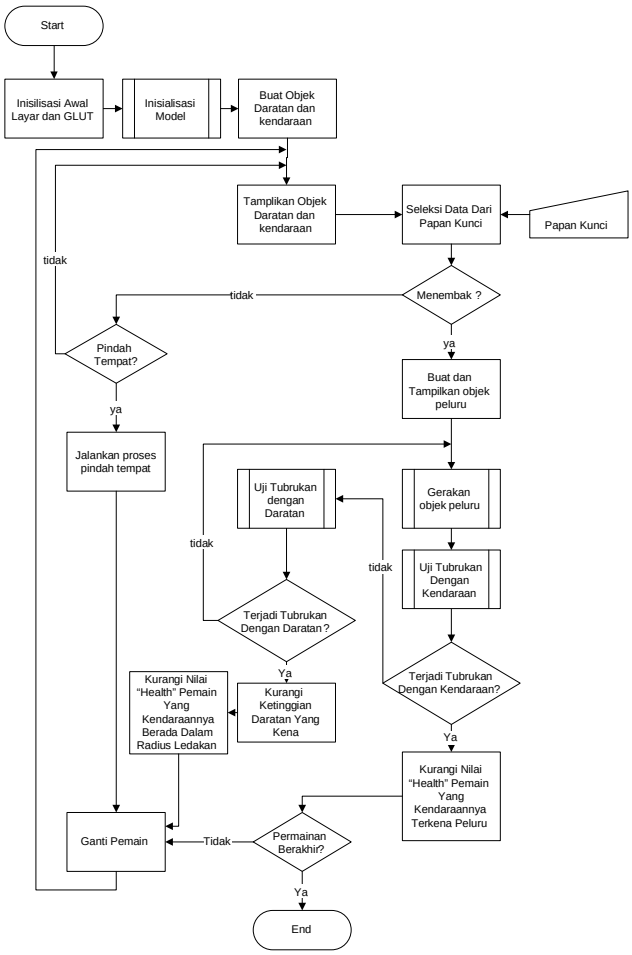
2. Apabila kedua garis tidak bertindihan maka AABB tidak berpotongan. Apabila ya, lakukan langkah diatas pada sumbu selanjutnya
3. Apabila pada ketiga sumbu tidak ada garis yang bertindihan maka AABB tidak berpotongan.

III. PERANCANGAN PERANGKAT LUNAK

Perancangan perangkat lunak dilakukan dengan menggunakan Diagram alir

2.1 Diagram Alir Program Utama

Diagram alir ini menggambarkan logika utama program, mulai dari pada saat program dimulai sampai program berakhir.

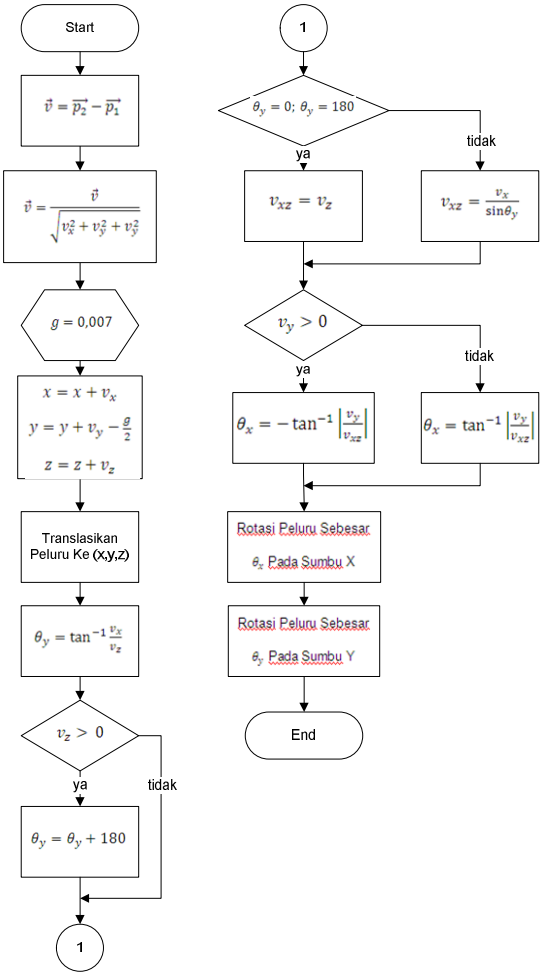


Gambar 3.1 Diagram Alir Program Utama

Diagram alir ini menggambarkan urutan dari proses proses yang dilakukan oleh sistem dari awal sampai akhir. Dari gambar diatas terlihat tata urutan jalannya program.

2.2 Diagram Alir Pergerakan Peluru

Sebelum digambar pada layar, terlebih dahulu peluru ditranslasi dan dirotasi sesuai dengan posisinya. Diagram alir pergerakan peluru tampak pada gambar dibawah ini.

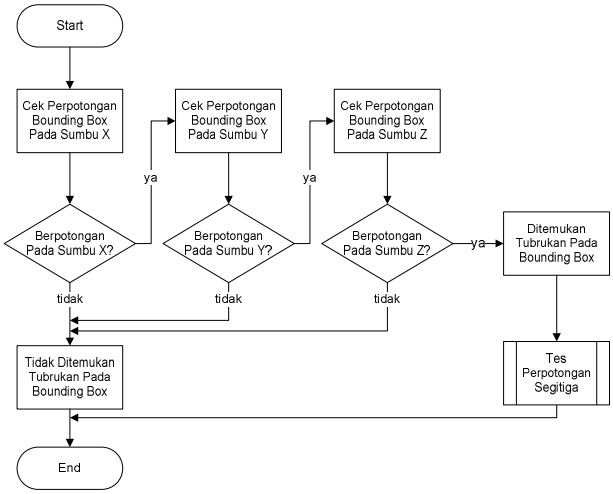


Gambar 3.2 Diagram Alir Proses Pergerakan Peluru

Diagram ini menggambarkan proses mencari posisi peluru, dan sudut rotasi peluru, sehingga peluru dapat digambarkan secara tepat.

2.3 Diagram Alir Deteksi Tubrukan

Gambar dibawah ini menunjukkan diagram alir dari proses deteksi tubrukan.

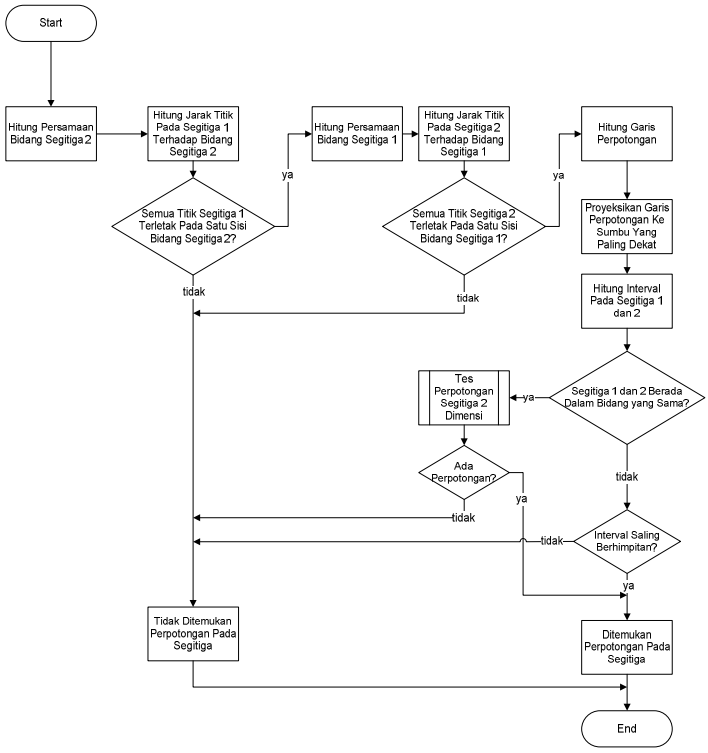


Gambar 3.3 Diagram Alir Deteksi Tubrukan

Deteksi Tubrukan digunakan untuk menentukan apakah peluru mengenai daratan, kendaraan, atau tidak mengenai apapun. Karena proses perhitungan tes perpotongan segitiga membutuhkan waktu yang lama, maka setiap objek memiliki sebuah *Bounding Volume*. Langkah pertama dari deteksi tubrukan adalah menghitung tabrakan antara *Bounding Volume*

2.4 Diagram Alir Tes Perpotongan Segitiga

Tes perpotongan segitiga adalah proses terakhir untuk menentukan apakah dua buah objek benar benar bertubrukan. Metode tes perpotongan segitiga yang digunakan dalam aplikasi ini mengacu pada metode *interval overlap* oleh Moller. Gambar dibawah ini mennunjukkan diagram alir tes perpotongan segitiga.

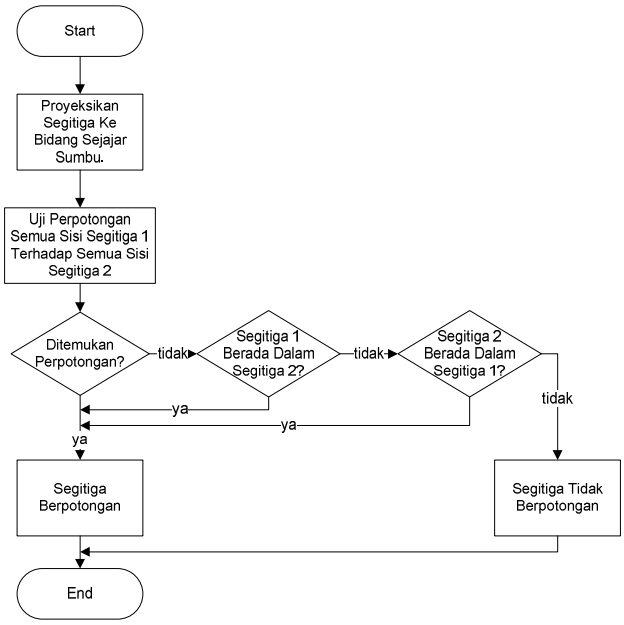


Gambar 3.4 Diagram Alir Tes Perpotongan Segitiga

2.5 Diagram Alir Tes Perpotongan Segitiga 2 Dimensi

Dalam aplikasi 3 dimensi, tetap diperlukan tes perpotongan segitiga 2 dimensi, tes ini digunakan untuk menangani keadaan dimana kedua segitiga yang di tes berada dalam bidang yang sama atau nyaris berimpit. Apabila kedua segitiga terletak pada bidang yang sama, maka tidak dapat dicari garis

perpotongan antar bidang seperti pada langkah tes perpotongan segitiga 3 dimensi, oleh karena itu keadaan seperti ini, walaupun jarang terjadi perlu ditangani secara khusus agar deteksi tubrukan tetap akurat. Dalam tes ini digunakan metode "*Faster Line Segment Intersection*" oleh Franklin Antonio untuk menemukan perpotongan antara dua buah sisi segitiga.



Gambar 3.5 Diagram Alir Tes Perpotongan Segitiga 2 Dimensi

IV. PENGUJIAN DAN ANALISIS

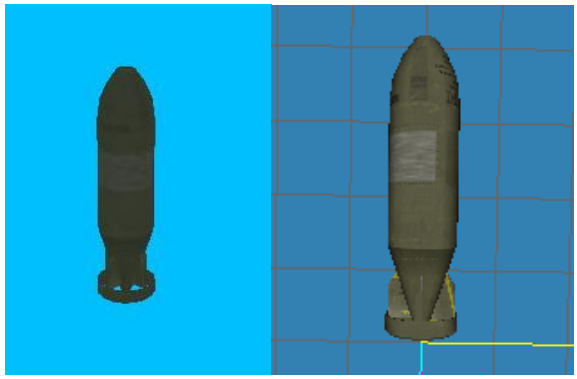
4.1 Implementasi dan Pengujian Proses Menampilkan Model Milkshape 3D ASCII

Pada pengujian menampilkan model Milkshape 3D ASCII menggunakan metode kotak hitam (*black box*) yang artinya aplikasi dihadapkan pada suatu kondisi kemudian melihat hasilnya. Tujuannya adalah untuk menguji tampilan pada aplikasi apakah sudah sesuai atau belum. Gambar 4.2 menunjukkan hasil tampilan pada aplikasi dan tampilan yang diharapkan, tampilan yang diharapkan merupakan tampilan yang berasal dari aplikasi Milkshape 3D yang digunakan untuk membuat model 3D tersebut.

Senarai untuk menampilkan model Milkshape ASCII 3D adalah:

```
AssignFile(MS3DFile, path+filename);
Reset(MS3DFile);
model.numShape:=StrToInt(StringReplace(temp,'Mes
hes: ','',[rfReplaceAll]));
setLength(model.shapes, model.numShape);

for j:=0 to model.numShape-1 do
begin
    ReadLn(MS3DFile,
model.shapes[j].numVertices);
setLength(model.shapes[j].vertices,model.shapes[
j].numVertices);
    ReadLn(MS3DFile,
model.shapes[j].numNormals);
setLength(model.shapes[j].Normals,model.shapes[j
].numNormals);
    ReadLn(MS3DFile,
model.shapes[j].numTriangles);
setLength(model.shapes[j].Triangles,model.shapes
[j].numTriangles);
ReadLn(MS3DFile,nFlags,model.shapes[j].triangles
[i].v[0],model.shapes[j].triangles[i].v[1],model
.shapes[j].triangles[i].v[2],model.shapes[j].tri
angles[i].n[0],model.shapes[j].triangles[i].n[1]
,model.shapes[j].triangles[i].n[2],nIndex);
end;
```



Gambar 4.1. Tampilan pada aplikasi Cannon 3D dan Tampilan pada aplikasi Milkshape 3D

Dari gambar diatas dapat dilihat bahwa pengujian menampilkan model Milkshape 3D ASCII sudah sesuai dengan yang diharapkan

4.2 Implementasi dan Pengujian Proses Pergerakan Peluru

Pergerakan peluru merupakan salah satu proses penting dalam aplikasi Cannon 3D. Proses ini mengatur pergerakan dan posisi peluru pada setiap *frame*. Sebelum peluru digerakkan, vektor arah gerak peluru dihitung terlebih dahulu dari posisi meriam dan variabel tenaga pemain.

Senarai yang digunakan untuk mencari vektor gerak peluru adalah:

```
long := (model.shapes[3].vertices[0].z-
model.shapes[1].vertices[0].z)*ModelScale;
if (Angle[1] > 270) or (Angle[1] < 90) then

GunBase[0]:=model.shapes[3].vertices[0].x*ModelS
cale
   else

GunBase[0]:=model.shapes[5].vertices[0].x*ModelS
cale;
   end;
GunBase[1]:=model.shapes[3].vertices[0].y*ModelS
cale;

XZ:=Long*Cos(Angle[0]*piover180);
Muzzle[1]:=
GunBase[1]+abs(Long*sin(Angle[0]*piover180));
Muzzle[0]:=
GunBase[0]+XZ*sin(Angle[1]*piover180);
Muzzle[2]:= XZ*cos(Angle[1]*piover180);

glSub(dir, Muzzle, GunBase);

Normalize(dir);
```

Senarai yang digunakan untuk menggerakkan peluru adalah:

```
position[0]:=position[0]+speed[0];
position[1]:=position[1]+speed[1]-0.0035;
position[2]:=position[2]+speed[2];
if Position[0]<-512 then Position[0]:=512;
if Position[2]<-512 then Position[2]:=512;
if Position[0]>512 then Position[0]:=-512;
if Position[2]>512 then Position[2]:=-512;
speed[1]:=speed[1]-0.007;
```

Pengujian pergerakan peluru dilakukan dengan membandingkan posisi peluru hasil perhitungan dan posisi peluru hasil dari keluaran sistem.
Masukan:
Posisi Awal = (0,7630 ; 55,2746 ; 474,1372).
Vektor Gerak Peluru = (0.0327 ; 0.8829 ; -0.4683).

Tabel 4.1 Hasil pengujian pergerakan peluru

Frame	Perhitungan			Keluaran Sistem		
	x	y	z	x	y	z
0	0,7630	55,2746	474,1372	0.7630	55.2746	474.1372
10	1,0900	63,0536	469,4542	1.0905	63.0541	469.4539
20	1,4170	70,1326	464,7712	1.4180	70.1336	464.7707
30	1,7440	76,5116	460,0882	1.7455	76.5130	460.0875
40	2,0710	82,1906	455,4052	2.0730	82.1925	455.4043
50	2,3980	87,1696	450,7222	2.4005	87.1720	450.7210
60	2,7250	91,4486	446,0392	2.7280	91.4514	446.0378
70	3,0520	95,0276	441,3562	3.0554	95.0309	441.3546
80	3,3790	97,9066	436,6732	3.3829	97.9103	436.6714
90	3,7060	100,0856	431,9902	3.7104	100.0898	431.9881
100	4,0330	101,5646	427,3072	4.0379	101.5692	427.3049

Dari tabel diatas terlihat bahwa hasil keluaran sistem sudah sesuai dengan hasil perhitungan. Perbedaan yang terjadi dikarenakan adanya pembulatan dalam perhitungan.

Pengujian rotasi peluru dilakukan dengan membandingkan rotasi peluru hasil perhitungan dan posisi peluru hasil dari keluaran sistem.

Tabel 4.2 Hasil pengujian rotasi peluru

Frame	Perhitungan		Keluaran Sistem	
	θ_x	θ_y	θ_x	θ_y
0	13,9991	14,0012	14.0000	14.0000
10	12,0374	14,0012	12.0384	14.0000
20	10,0466	14,0012	10.0477	14.0000
30	8,0311	14,0012	8.0322	14.0000
40	5,9953	14,0012	5.9965	14.0000
50	3,9443	14,0012	3.9455	14.0000
60	1,8830	14,0012	1.8843	14.0000
70	-0,1831	14,0012	-0.1818	14.0000
80	-2,2487	14,0012	-2.2473	14.0000
90	-4,3085	14,0012	-4.3071	14.0000

Dari tabel diatas terlihat bahwa hasil keluaran sistem sudah sesuai dengan hasil perhitungan. Perbedaan yang terjadi dikarenakan adanya pembulatan dalam perhitungan.

4.3 Implementasi dan Pengujian Deteksi Tubrukan

Deteksi tubrukan berfungsi untuk mendeteksi bila objek peluru mengenai objek pemain atau objek daratan. Senarai untuk pengujian tubrukan antar *Bounding Volume* adalah:

```
result:=true;
For i:=0 to 2 do
    If (MIN[i]>colidee^.MAX[i]) or
(colidee^.MIN>MAX[i]) then
          begin
               result:=false;
               break;
          end;
```

Apabila terdeteksi adanya tubrukan pada *Bounding Volume*, maka barulah diuji apakah ada segitiga dalam *Bounding Volume* yang berpotongan. Untuk pengujian ini digunakan metode *interval overlap* yang dikembangkan oleh Moller. Senarai untuk pengujian interseksi segitiga adalah:

```
NewComputeInterval(vp0,vp1,vp2,dv0,dv1,dv2,dv0dv
1,dv0dv2,a,b,c,x0,x1);
NewComputeInterval(vp0,vp1,vp2,dv0,dv1,dv2,dv0dv
1,dv0dv2,a,b,c,x0,x1);
if (notcoplanar=false) then
result:=CoplanarTriTri(N1, tri1[0], tri1[1],
tri1[2], tri2[0], tri2[1], tri2[2])
else
    isect2[0]:=tmp+e*xx*y1;
    isect2[1]:=tmp+f*xx*y0;

    SortDesc (isect1[0],isect1[1]);
    SortDesc (isect2[0],isect2[1]);

    if ((isect1[1]<isect2[0]) or
(isect2[1]<isect1[0])) then
       result:=false
    else
       result:=true;
```

Dalam aplikasi ada kemungkinan bahwa segitiga yang diuji ternyata berada dalam bidang yang sama, untuk menangani kondisi ini, ditambahkan proses tes perpotongan segitiga 2 Dimensi. Senarai dari tes perpotongan segitiga 2 Dimensi adalah:

```
if EdgeAgainstTriEdges(V0, V1, U0, U1, U2, i0,
i1) = true then
   result:=true
else
   if EdgeAgainstTriEdges(V1, V2, U0, U1, U2,
i0, i1) = true then
      result:=true
   else
      if EdgeAgainstTriEdges(V2, V0, U0, U1, U2,
i0, i1) = true then
         result:=true
       else
         if PointInTri(V0, U0, U1, U2, i0, i1) =
true then
            result:=true
         else
            if PointInTri(U0, V0, V1, V2, i0,
i1) = true then
               result:=true
            else
```

Untuk pengujian diberikan beberapa variasi perhitungan, antara peluru, daratan dan kendaraan. Hasil dari pengujian ditunjukkan pada tabel dibawah ini:

Tabel 4.3 Hasil pengujian deteksi tubrukan

Objek Diuji	**Jumlah Segitiga**	**Jumlah Pengujian Per-*frame***	**FPS**
1 Peluru, 1 Daratan	521 dan 128	66.688	41,7
1 Peluru, 2 Daratan	521 dan 256	133.376	27,0
1 Peluru, 3 Daratan	521 dan 384	200.064	21,7
1 Peluru, 4 Daratan	521 dan 512	266.752	16,7
1 Peluru, 1 Kendaraan	521 dan 1.203	626.763	7,6
1 Peluru, 1 Daratan, 1 Kendaraan	521 dan 1.331	693.451	7,3
1 Peluru, 2 Daratan, 1 Kendaraan	521 dan 1.459	760.139	6,6
1 Peluru, 3 Daratan, 1 Kendaraan	521 dan 1.587	826.827	6,2
1 Peluru, 4 Daratan, 1 Kendaraan	521 dan 1.715	893.515	5,7
Tanpa *Bounding Volume*	521 dan 133.478	69.542.038	0,1

Dari tabel diatas dapat dilihat bahwa semakin besar jumlah pengujian per-*frame* maka besarnya fps semakin berkurang. Tanpa bounding volume jumlah tes metode interval overlap yang dilakukan oleh sistem menjadi sangat besar, sehingga akan menghambat kinerja keseluruhan sistem, dan sistem akan berjalan dengan sangat lambat.

V. PENUTUP

5.1 Kesimpulan

Dari hasil pengujian dan analisis maka dapat disimpulkan hal-hal sebagai berikut.

1. Deteksi tubrukan pada aplikasi ini dibagi menjadi 2 tingkat, yaitu Tes tubrukan *Bounding Volume* dan tes perpotongan segitiga.
2. *Bounding Volume* digunakan untuk mengurangi jumlah tes perpotongan segitiga yang dilakukan pada setiap *frame*.
3. Tes perpotongan segitiga dilakukan untuk mengetahui apakah kedua objek berpotongan.
4. Tes perpotongan segitiga 2 dimensi dilakukan untuk menangani kondisi dimana kedua segitiga berada pada bidang yang sama.
5. Dengan variasi perhitungan sampai 893.515 tes perpotongan segitiga, aplikasi masih dapat mendeteksi adanya tubrukan pada objek.
6. Tanpa *Bounding Volume* aplikasi berjalan dengan sangat lambat.

5.1 Saran

Berdasarkan pengujian terhadap Permainan Cannon 3D yang telah dibuat, dapat diberikan beberapa saran sebagai berikut.

1. Aplikasi permainan Cannon 3D ini dapat dikembangkan lebih lanjut dengan permainan untuk satu pemain. Dimana pemain dapat melawan algoritma yang memiliki tingkat kesulitan berbeda – beda.
2. Aplikasi dapat dikembangkan lebih lanjut agar dapat dimainkan oleh beberapa pemain secara bersamaan dalam jaringan.
3. *Bounding Volume* pada kendaraan pemain dapat dipecah menjadi bagian bagian yang lebih kecil untuk mendapatkan performa yang lebih baik.

DAFTAR PUSTAKA

[1] Moller, T., *Fast Triangle-Triangle Intersection Test,* Journal of Graphics Tools, 1997.

[2] Astle, D. and Kevin Hawkins, *Beginning OpenGL Game Programming*, Premier Press, Massachusetts, 2004.

[3] Martz, P., *OpenGL Distilled*, Addison Wesley Professional, Massachusetts, 2006.

[4] Wright, Richard S. Jr and Benjamin Lipchak, *OpenGL SuperBible, Third Edition,* Sams Publishing, United States of America, 2004.

[5] Moller, T. and Eric Haines, *Real-Time Rendering Second Edition,* A K Peters, Massachusetts, 2002.

[6] Glassner, Andrew S, *Graphics Gems 3*, Academic press, California, 1995.

[7] ---, www.lighthouse3d.com/opengl, April 2007.

[8] ---, nehe.gamedev.net, April 2007.

[9] ---, www.opengl.org/resources/libraries/glut, April 2007.

[10] ---, www.glprogramming.com/blue, April 2007.

[11] ---, www.glprogramming.com/red, April 2007.

BIOGRAFI PENULIS

Fitriadi Nugroho, lahir di Semarang, Jawa Tengah, 9 Juni 1986. Menempuh pendidikan di SDN Jrakah 5 Semarang, SLTP Negeri 1 Semarang, dan SMU Negeri 3 Semarang, saat ini sedang menyelesaikan pendidikan program Strata 1 Jurusan Teknik Elektro Universitas Diponegoro, mengambil konsentrasi Informatika dan Komputer.

Menyetujui dan Mengesahkan,

Pembimbing I,

Aghus Sofwan, S.T., M.T.
NIP. 132 163 757
Tanggal ……...

Pembimbing II,

Agung BP, ST, MIT
NIP. 132 137 932
Tanggal ……