**TUGAS BESAR GRAFKOMP**

**PROGRAM ANIMASI 3D PERKEMAHAN**

**DENGAN OPENGL CODEBLOCKS**

Diajukan Untuk Memenuhi

Salah Satu Tugas Mata Kuliah Komputer Grafika

Dosen : Hendri Karisma, S. Kom.



Disusun Oleh : Ridzkin Panca F. - (10108394)  
 Zumara Ibrahim A.P. - (10108397)  
 Rifqi Febriwilliandra - (10108402)  
 Tri Nuzul Satrianto - (10108403)

Kelas : IF-8/S1/VIII

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK DAN ILMU KOMPUTER**

**UNIVERSITAS KOMPUTER INDONESIA**

**2012**

# DAFTAR ISI

DAFTAR ISI 2

KATA PENGANTAR 4

ABSTRAK 5

BAB I 6

Pendahuluan 6

1.1 Latar Belakang Masalah…………………………………………………………………...

1.2 Maksud dan Tujuan………………………………………………………………………...

1.3 Batasan Masalah……………………………………………………………………………

BAB II 8

LANDASAN TEORI 8

2.1 Pengertian Grafika Komputer 8

2.2 Elemen Dasar Grafika 8

2.2.1. Point 8

2.2.2. Polyline 9

2.2.3 Polygon 9

2.2.4 Filled Polygon (Face) 9

2.2.5. Gradate Polygon 9

2.3 Grafik Komputer 2D 9

2.4 Grafik komputer 3D 10

2.5 Perbedaan Grafik 2D dan 3D 10

2.6 Transformasi 10

2.7 Color (Warna) 13

2.8 Lighting (pencahayaan) 14

2.9 Bayangan 15

2.10 Blending (Pencampuran) 15

2.11 Texture Mapping 16

2.10.1 Menentukan Tekstur Image : 16

2.10.2. Mendefinisikan gambar sebagai sebuah teksture 17

2.10.3. Mengubah gambar tekstur : 17

2.10.4. Mapping Tekstur : 17

2.12 Fog 18

2.13 Pengertian OpenGl 18

2.14 Sejarah OpenGL 18

2.15 Konfigurasi OpenGL pada Codeblocks 20

BAB III 22

IMPLEMENTASI PROGRAM 22

Bab IV 81

Kesimpulan dan saran 81

# KATA PENGANTAR

Sebelumnya kami mengucapkan puji dan syukur kehadirat TUHAN YANG MAHA ESA yang telah melimpahkan rahmat dan karunianya kepada kami, sehingga kami dapat menyelesaikan Tugas Grafika Komputer ini tepat pada waktunya.

Terdorong oleh rasa ingin tahu, kemauan, kerja sama dan kerja keras, kami kerahkan seluruh upaya demi mewujudkan ke ingin-tahuan dan tugas dari dosen pembimbing untuk menyelesaikan tugas ini. Semoga tulisan ini dapat memenuhi kewajiban kami dalam tugas perkuliahan Grafika Komputer.

Adapun harapan kami, semoga tulisan ini dapat menambah wawasan pembaca mengenai Grafika Komputer dan penerjemah bahasa pencitraan OpenGL, dengan maksud nantinya pembaca mampu untuk memahami apa itu Grafika Komputer, OpenGL, dan lain sebagainya yang menyangkut objek 3 dimensi (3D).

Kami menyadari bahwa tulisan ini masih jauh dari sempurna, untuk itu kami mengharapkan kritik dan saran yang membangun dari pembaca demi kesempurnaan tulisan ini.

Bandung, 1 Juni 2012

Hormat Kami,

Tim Penulis

# ABSTRAK

Gambar 3D (3 dimensi) seringkali kita dengar sewaktu kita masih kecil dulu, namun tidak sedikit orang yang tidak tahu bagaimana mengimplementasikan suatu objek menjadi sesuatu yang terlihat nyata dan megah. Seiring berjalannya waktu, dampak perkembangan teknologi sangat terasa bagi kita bagaimana mudahnya membuat sesuatu yang kita bayangkan menjadi nyata dalam hal ini yang dimaksud yaitu membuat suatu objek 3D.

Begitu banyak teknologi/perangkat lunak untuk mengimplementasikan apa yang kita bayangkan menjadi sesuatu yang dapat kita ciptakan kedalam bentuk 3 dimensi (3D), diantaranya yaitu OpenGL. OpenGL(*Open Graphics Library*) adalah suatu library grafis yang digunakan untuk keperluan-keperluan pemrograman grafis. Spesifikasi standar yang dimiliki oleh library ini mendefinisikan sebuah cross-bahasa, cross-platform API untuk menulis aplikasi komputer dalam bentuk 2D dan 3D grafis. OpenGL bersifat open source, dapat dipakai pada banyak platform (Windows ataupun Linux) dan dapat digunakan pada berbagai jenis compiler bahasa pemrograman seperti C++, Delphi, Java ataupun VB.

Disini kami menggunakan codeblocks dan OpenGL sebagai sarana untuk mengimplementasikan animasi yang kami bangun yaitu “Perkemahan 3D”. Dimana terdapat Objek-objek yang berupa :

1. Pegunungan,
2. Pohon yang beraneka jenis,
3. Bebek yang sedang berjalan-jalan,
4. Meja,
5. Kursi,
6. Tempat perapian,
7. Matras
8. Rumah, dan
9. Sungai yang mengelilingi pegunungan.

# BAB I

# Pendahuluan

1. **Latar Belakang Masalah**

Semakin berkembangnya ilmu pengetahuan dan teknologi grafis pada masa sekarang ini, maka pengetahuan mengenai grafik komputer semakin banyak dipakai. Namun sebelum kita membahas lebih jauh mengenai grafik komputer, kita akan mengetahui dahulu definisi sebenarnya tentang apa itu grafik komputer.

Secara umum grafik komputer adalah gambar yang dihasilkan oleh komputer, yang hasilnya sudah sering kita lihat seperti di majalah dan televisi. Disini dibahas bahwa tiap karakter yang dihasilkan diambil dari library dari bentuk karakter yang ada pada memori komputer. Gambar-gambar yang berada pada majalah atau televisi tersebut ada beberapa yang terlihat sangat natural, sehingga kita para pembaca akan sulit membedakan mereka buatan atau hasil dari fotografi asli.

Selain itu grafik komputer juga digunakan untuk membuat design dari background ataupun objek-objek dalam game di komputer. Gambar-gambar yang ada pada game itu adalah gabungan antara kenyataan dan imajinasi dari programmer-nya. Dalam bidang lain, grafik komputer digunakan dalam dunia seni, entertainment, dan publikasi. Khususnya dipakai dalam produksi film, animasi, dan spesial efek. Animasi dibuat dengan membuat sebuah gambar yang berkelanjutan di film atau videotape. Gambar satu dan yang selanjutnya hanya mempunyai perbedaan yang tipis (slightly different). Bidang-bidang lain yang menggunakan grafika komputer adalah untuk browsing di World Wide Web, untuk pengawasan proses, dan banyak lagi.

Dari penjelasan diatas penulis mencoba untuk membuat animasi yang berupa objek 3D, adapun animasi yang akan dibangun yaitu Pemandangan 3D. Objek-objek yang akan dibangun yaitu :

1. Pegunungan,
2. Pohon yang beraneka jenis,
3. Bebek yang sedang berjalan-jalan,
4. Meja,
5. Kursi,
6. Tempat perapian,
7. Matras
8. Rumah, dan
9. Sungai yang mengelilingi pegunungan.
10. **Maksud dan Tujuan**

**Maksud**

Maksud dari penulisan ini adalah untuk membuat animasi pemandangan 3D dengan menggunakan OpenGL.

**Tujuan**

Adapun tujuan yang akan dicapai dari penulisan laporan ini adalah sebagai berikut :

1. Menampilkan simulasi keadaan pemandangan dari segala sisi.
2. Mengetahui fungsi-fungsi pada openGL.
3. **Batasan Masalah**

Adapun batasan masalah dalam pembuatan animasi 3D ini adalah sebagai berikut :

1. Bahasa yang digunakan bahasa pemrograman C++
2. IDE yang digunakan codeblocks dengan openGL
3. Program ini hanya menampilkan objek yang bertema perkemahan 3D.

# BAB II

# LANDASAN TEORI

## Pengertian Grafika Komputer

Grafika komputer (*Computer graphics*) adalah bagian dari [ilmu komputer](http://id.wikipedia.org/wiki/Ilmu_komputer) yang berkaitan dengan pembuatan dan manipulasi gambar secara digital. Bentuk sederhana dari grafika komputer adalah [grafika komputer 2D](http://id.wikipedia.org/w/index.php?title=Grafika_komputer_2D&action=edit&redlink=1) yang kemudian berkembang menjadi [grafika komputer 3D](http://id.wikipedia.org/wiki/Grafika_komputer_3D), [pemrosesan citra](http://id.wikipedia.org/w/index.php?title=Pemrosesan_citra&action=edit&redlink=1) *(image processing),* dan [pengenalan pola](http://id.wikipedia.org/wiki/Pengenalan_pola) *(pattern recognition).* Grafika komputer sering dikenal juga dengan istilah [visualisasi](http://id.wikipedia.org/wiki/Visualisasi) data.

Bagian dari grafika komputer meliputi:

1. [Geometri](http://id.wikipedia.org/wiki/Geometri): mempelajari cara menggambarkan permukaan bidang
2. [Animasi](http://id.wikipedia.org/wiki/Animasi): mempelajari cara menggambarkan dan memanipulasi gerakan
3. [Rendering](http://id.wikipedia.org/w/index.php?title=Rendering&action=edit&redlink=1): mempelajari [algoritma](http://id.wikipedia.org/wiki/Algoritma) untuk menampilkan efek cahaya
4. [Citra](http://id.wikipedia.org/wiki/Citra) (Imaging): mempelajari cara pengambilan dan penyuntingan gambar.

Teknik-teknik yang dipelajari dalam grafika komputer adalah teknik-teknik bagaimana membuat atau menciptakan gambar menggunakan komputer. Ada perbedaan yang sangat mendasar antara foto dan gambar, yaitu pada foto semua detail obyek terlihat sedangkan pada gambar (baik itu gambar manusia atau gambar komputer) tidak dapat memperlihatkan semua detail yang ada tetapi hanya detail-detail yang dianggap penting dalam menunjukkan pola suatu gambar.

## Elemen Dasar Grafika

Ada beberapa elemen dasar dari grafika komputer antara lain:

1. Point
2. Polyline
3. Polygon
4. Filled Polygon ( Face )
5. Gradate Polygon

### 2.2.1. Point

*Point* adalah sebuah titik yang digunakan untuk membangun obyek. Setiap titik dalam obyek 3 dimensi memiliki nilai dalam x, y dan z.

### 2.2.2. Polyline

*Polyline* adalah sebuah fungsi yang dibentuk dari beberapa garis yang saling berhubungan dan membentuk sebuh kurva yang terbuka.

### 2.2.3 Polygon

*Polygon* adalah suatu fungsi yang mirip dengan polyline hanya saja hasilnya adalah kurva tertutup, sedangkan *polyline* hasilnya kurva terbuka.

### 2.2.4 Filled Polygon (Face)

*Filled Polygon* adalah sebuah *polygon* yang bagian dalamnya diwarnai atau dipenuhi dengan sebuah warna tertentu. *Filled polygon* biasanya digunakan sebagai *face* dari pembentukan obyek–obyek 3 Dimensi.

### 2.2.5. Gradate Polygon

*Gradate polygon* adalah sebuah *polygon* yang bagian dalamnya memiliki warna – warna yang bergradasi dari satu warna ke warna yang lainnya.

## Grafik Komputer 2D

Grafik komputer 2D adalah pembuatan objek gambar dengan menggunakan 2 titik sebagai acuannya yaitu sumbu x dan y. Grafik 2D ini dapat digunakan dalam berbagai aplikasi yang pada awalnya dikembangkan pada teknologi cetak tradisional dan gambar, seperti tipografi, kartografi, gambar teknik, iklan, dan lain-lain.

Grafik komputer 2D ini merupakan langkah paling awal dalam membentuk model objek yang akan dibangun dalam grafik komputer 3D. Dalam aplikasi, gambar dua dimensi adalah bukan hanya representasi dari objek dunia nyata, tetapi sebuah artefak independen dengan nilai tambah semantik. Keseluruhan obyek 2D dapat dimasukkan dengan jumlah lebih dari satu, model yang akan dibentuk sesuai dengan kebutuhan. Tahap rekayasa hasil obyek 2D dapat dilakukan dengan aplikasi program grafis seperti Adobe Photoshop, Corel Draw, dan lain sebagainya.

## Grafik komputer 3D

Grafik komputer 3D merupakan representasi dari data geometrik 3 dimensi sebagai hasil dari pemrosesan dan pemberian efek cahaya terhadap grafik komputer 2D. hasilnya dapat ditampilkan secara real time untuk keperluan simulasi. Prinsip yang dipakai mirip dengan grafik komputer 2D dalam penggunaan algoritma, grafika vektor, model frame kawat (wire frame model), dan grafik rasternya.

Grafik komputer 3D sering disebut sebagai model 3D. Namun, model 3D ini lebih menekankan pada representasi matematis untuk objek 3 dimensi. Obyek pada grafik 3D adalah sekumpulan titik-titik 3D (x,y,z) yang membentuk suatu face (bidang) yang digabungkan menjadi satu kesatuan. Face sendiri adalah gabungan titik-titik yang membentuk bidang tertentu. Data matematis ini belum bisa dikatakan sebagai gambar grafis hingga saat ditampilkan secara visual pada layar komputer atau printer. Proses penampilan suatu model matematis ke bentuk citra 2 D biasanya dikenal dengan proses 3D rendering.

## Perbedaan Grafik 2D dan 3D

Perbedaan yang paling mendasar dan terlihat dengan sangat jelas adalah tampilan gambarnya. Gambar 2D tampil flat adn frame tampilannya cenderung terbatas karena objek gambarnya disajikan hanya dengan sumbu x dan y. Sedangkan pada grafik 3D, gambar yang ditampilkan lebih hidup, membentuk ruang, tidak flat, serta framenya lebih luas yang dikarenakan gambar 3D disajikan dengan 3 sumbu, yaitu x, y, dan z.

## Transformasi

Transformasi dasar pada objek dua dimensi yang pertama adalah translasi (*translation*). Translasi berarti memindahkan suatu objek sepanjang garis lurus dari suatu lokasi koordinat tertentu ke lokasi yang lain. Transformasi skala (*scaling)* digunakan untuk mengubah ukuran suatu objek, sedangkan rotasi (*rotation)* adalah pemindahan objek menurut garis melingkar.

Jenis jenis dari transformasi dasar:

1. Translasi

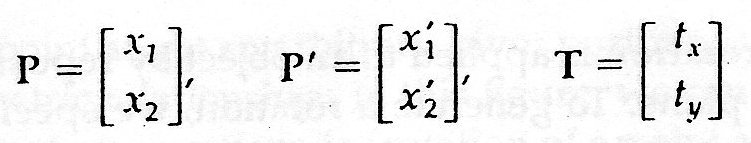
Translasi dilakukan dengan penambahan translasi pada suatu titik koordinat dengan translasi vektor atau shift vektor, yaitu (tx,ty), dimana tx adalah translation vektor menurut sumbu x, sedangkan ty adalah translation vektor menurut sumbu y. koordinat baru titik yang ditranslasi dapat diperoleh dengan

X’= x + tx

Y’= y + ty

Dimana (x,y) adalah koordinat asal suatu objek dan (x’ , y’) adalah koordinat baru objek tersebut setelah ditranslasi.

Kadang-kadang transformasi dinyatakan dalam bentuk matriks, sehingga matriks tranformasi untuk translasi dapat dinyatakan sebagai berikut :



Dengan demikian translasi dua dimensi dapat ditulis dalam bentuk matriks :

|  |
| --- |
| P’ = P + T |

Disamping dinyatakan dalam vektor kolom, matriks transformasi dapat dituliskan dalam bentuk vektor baris, sehingga menjadi P = [ x y ] dan T = [ tx ty ]. Bentuk vektor kolom adalah standar dari symbol matematik, yang juga berlaku bagi notasi grafik seperti GKS dan PHIGS.

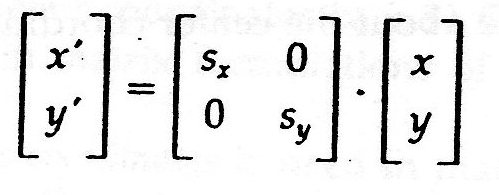
1. Skala

Transformasi skala adalah perubahan ukuran suatu objek. Koordinat baru dapat diperoleh dengan melakukan perkalian nilai koordinat dengan scaling factor, yaitu (sx , sy) ,dimana sx adalah scaling factor menurut sumbu x, sedangkan sy adalah scaling factor menurut sumbu y. koordinat baru titik yang diskala dapat diperoleh dengan

X’= x + sx

Y’ = y + sy

Dimana (x , y) adalah koordinat asal suatu objek dan (x’,y’) adalah koordinat setelah diskala. Matriks transformasi untuk skala dapat dinyatakan sebagai berikut:



Dengan demikian skala dapat juga dituliskan

P’ = S . P

Scaling factor sx dan sy dapat diberikan sembarang nilai positif. Nilai lebih dari 1 menyebabkan objek diperbesar, sebaliknya bila nilai lebih kecil dari 1, maka objek akan diperkecil. Bila sx dan sy mempunyai nilai yang sama, maka skala disebut uniform scaling. Nilai yang tidak sama dari sx dan sy menghasilkan differential scaling, yang biasa digunakan pada program aplikasi.

1. Rotasi

Rotasi dua dimensi pada suatu objek kan memindahkan objek tersebut menurut garis melingkar. Pada bidang xy. Untuk melakukan rotasi diperlukan sudut rotasi θ dan pivot point (xp’ yp ) atau rotasi point dimana objek di rotasi, seperti pada gambar 5-3 nilai positif dari sudut rotasi menentukan arah rotasi berlawanan dengan jarum jam, dan sebaliknya nilai negative akan memutar objek searah jarum jam.

Rotasi dapat dilakukan dengan pivot point yaitu titik pusat koordinat, seperti pada gambar 5-4. Pada betuk ini, r adalah jarak konstan dari titik pusat, sudut φ adalah sudut posisi suatu titik dengan sumbu horizontal, sedangkan θ adalah sudut rotasi. Menggunakan trigonometri, transformasi dapat dinyatakan dengan sudut θ dan φ sebagai berikut:

X’= r cos (φ + θ) = r cos φ cos θ – r sin φ sin θ

y’ = r sin (φ + θ) = r cos φ sin θ + r sin φ cos θ

sedangkan dengan koordinat polar diketahui bahwa

x = r cos φ, y = r sin φ

dengan melakukan substitusi, diperoleh rumus transformasi untuk rotasi suatu titik (x, y) dengan sudut rotasi θ sebagai berikut:

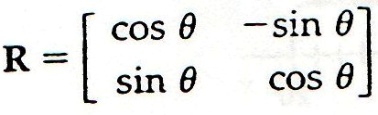
x’ = x cos θ – y sin θ

y’ = x sin θ – y cos θ

matriks transformasi untuk rotasi dapat dinyatakan sebagai berikut:

P’ = R . P

Rotasi dapat dinyatakan dalam bentuk lain, yaitu matriks. Matriks rotasi dapat dituliskan dengan



Rotasi suatu titik terhadap pivot point (xp’ yp ) seperti pada gambar 5-5, menggunakan bentuk trigonometri, secara umum dapat dituliskan sebagai berikut:

X’ = xp +(x - xp) cos θ – (y - yp) sin θ

Y’ = yp + (x – xp) sin θ + (y – yp) cos θ

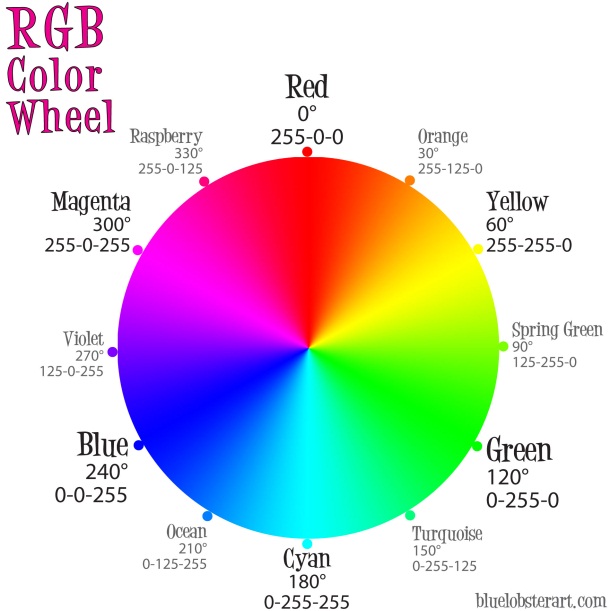
## Color (Warna)

Bentuk gelombang elektromagnetik yang terkandung dalam cahaya yang berasal dari sumber cahaya. Spectrum warna memiliki panjang gelombang elektomagnetik antara 350-750 nanometer .

**Pembagian Warna**

* RGB (Red-Green-Blue) : warna dasar yang dijadikan patokan warna secara universal (primary colors)
* CMYK (Cyan-Magenta-Yellow-Black) : Sistem representasi pada warna tinta yang diterapkan dalam dunia fotografi dan produksi grafika

Sistem Warna Lingkaran (the color wheel) : Digunakan untuk mengkombinasikan dan mengharmonikan warna pada karya seni dan design



Gambar 2.0 Pembagian warna

Warna dapat didefinisikan secara obyektif/fisik sebagai sifat cahaya yang dipancarkan, atau secara subyektif/psikologis sebagai bagian dari pengalaman indera pengelihatan. Secara obyektif atau fisik, warna dapat diberikan oleh panajang gelombang. Dilihat dari panjang gelombang, cahaya yang tampak oleh mata merupakan salah satu bentuk pancaran energi yang merupakan bagian yang sempit dari gelombang elektromagnetik.

Dari sekian banyak warna, dapat dibagi dalam beberapa bagian yang sering dinamakan dengan sistem warna Prang System yang ditemukan oleh Louis Prang pada 1876 meliputi :

1. Hue, adalah istilah yang digunakan untuk menunjukkan nama dari suatu warna, seperti merah, biru, hijau dsb.
2. Value, adalah dimensi kedua atau mengenai terang gelapnya warna. Contohnya adalah tingkatan warna dari putih hingga hitam.
3. Intensity, seringkali disebut dengan chroma, adalah dimensi yang berhubungan dengan cerah atau suramnya warna.

## Lighting (pencahayaan)

Lighting merupakan proses menghitung intensitas cahaya terutama pada 3-Dimensi point, biasanya diatas suatu permukaan.

Beberapa cara mengatasi masalah pencahayaan, antara lain :

1. Mengerti persepsi dari cahaya (warna)
2. Membuat sebuah solusi untuk merepresentasikan dan menghasilkan warna menggunakan komputer.
3. Mengerti akan pengaruh cahaya dan objek

## Bayangan

1. Bayangan akan muncul saat cahaya jatuh menyinari suatu objek.
2. Pada dunia maya, layaknya cahaya, terdapat beberapa jenis bayangan yang dapat dihasilkan oleh komputer.

Bayangan bekerja sama dengan cahaya untuk memberi kesan natural atau realistic pada scene yang ada. Bayangan dapat membantu mendefinisikan posisi objek-objek, apakah berada di lantai atau melayang di udara. Bayangan yang dihasilkan bisa tajam dan solid namun bisa juga lembut dan buram (blurry). Keberadaan bayangan atau ketiadaannya dapat digunakan untuk memberi keseimbangan dan kontras pada objek-objek di dalam scene.

## Blending (Pencampuran)

Pencampuran merupakan fungsi yang menggabungkan nilai warna dari sumber dan tujuan. Operasi campuran yaitu cara yang paling alami untuk mengetahui bahwa komponen RGB adalah suatu fragmen yang mewakili warna dan komponen alfa adalah suatu fragmen yang mewakili sifat tidak tembus cahaya.

**Faktor sumber dan tujuan**

Pada proses pencampuran, nilai cairan warna yang masuk fragmen (sumber) digabungkan dengan warna yang sesuai dengan nilai saat ini yang disimpan pada piksel (tujuan) dalam dua tahap proses. Yang pertama menghitung faktor sumber dan tujuan, factor-faktor tersebut adalah RGBA quadruplets yang masing-masing dikalian dengan komponen-komponen R, G, B dan nilai-nilai dari sumber dan tujuan. Kemudian komponen yang sesuai dalam dua set RGBA quadruplets. Secara sistematis, faktor sumber dan tujuan pencampuran (SR, Sg, Sb, Sa) dan (Dr, Dg, dB, Da) dan nilai RGBA ditandai dengan s atau d dan terakhir nilai RGBA dicampurkan yang diperoleh dengan (RsSr + RdDr, GsSg + GdDg, BsSb + BdDb, AsSa + Adda) dimana setiap komponen adalah quadruplets is eventually clamped to [0,1].

Dengan menggunakan glBlendFunc () untuk persediaan pada dua hal utama, yang pertama menentukan bagaimana faktor sumber dan tujuan harus dihitung dan yang kedua menunjukan bagaimana faktor sumber dan tujuan dihitung. Dan untuk proses pencampurannya harus ada faktor pengaktifannya menggunakan : glEnable (GL\_BLEND). Menggunakan **glDisable ()** dengan GL\_BLEND untuk menonaktifkan Pencampuran dan menggunakan konstan GL\_ONE (sumber) dan GL\_ZERO (tujuan) memberikan hasil yang sama seperti ketika Pencampuran dinonaktifkan. Nilai-nilai ini bersifat default dengan void **glBlendFunc** (GLenum *sfactor,* GLenum *dfactor)*.

Mengontrol bagaimana nilai warna dalam fragmen yang diproses digabungkan dengan yang sudah disimpan dalam framebuffer (tujuan). Pendapat *sfactor* menunjukkan bagaimana untuk menghitung faktor sumber Pencampuran dan *dfactor* menunjukkan bagaimana untuk menghitung faktor tujuan Pencampuran. Campuran faktor yang diasumsikan terletak pada rentang [0,1]; setelah nilai warna dalam sumber dan tujuan digabungkan, setelah dihitung kisaran [0,1].

## Texture Mapping

Texture mapping merupakan teknik pemetaan sebuah tekstur pada pola gambar wireframe, dimana wireframe yang telah dibuat akan ditampilkan memiliki kulit luar seperti tekstur yang diinginkan. Dalam pemberian tekstur, perlu diperhatikan dasarnya seperti:

1. Menentukan tekstur
2. Membaca atau membangkitkan tekstur
3. Menandai tekstur
4. Mengenablekan tekstur
5. Menandai koordinat tekstur pada vertek
6. Menentukan parameter tekstur
7. Wrapping , filtering, dsb.

Langkah-langkah dalam memulai mapping sebuah tekstur yakni dengan spesifikasi dibawah ini :

## 2.10.1 Menentukan Tekstur Image :

1. Mendefinisikan tekstur image dari sebuah array teksel (element tekstur ) ke dalam memory cpu : Glubyte my\_texels[512][512];
2. Mendefinisikan seperti semua peta piksel yang lain
3. Gambar yang didefinisikan (baik secara manual maupun dengn suatu fungsi matematik tertentu).
4. Membangkitkan dengan kode aplikasi
5. Mengenablekan tekstur mapping
6. glEnable(GL\_TEXTURE\_2D)
7. OpenGL mendukung 1 sampai 4 dimensional tekstur mapping

## 2.10.2. Mendefinisikan gambar sebagai sebuah teksture

glTexImage2D(target,level,components,w,h,border,format,type, texels );

Keterangan :

1. target: tipe dari teksture, e.g. GL\_TEXTURE\_2D
2. level: digunakan untuk *mipmapping*
3. components: element per texel
4. w, h: lebar dan tinggi dari texels pada pixels
5. border: digunakan untuk smoothing
6. format and type: menjelaskan texels
7. texels: pointer ke array texel
8. glTexImage2D(GL\_TEXTURE\_2D, 0, 3, 512, 512, 0,GL\_RGB, GL\_UNSIGNED\_BYTE, my\_texels);

## 2.10.3. Mengubah gambar tekstur :

1. OpenGL meminta dimensi tekstur untuk menjadi dasar dari 2
2. Jika dimensi dari image bukan power ke 2, gluScaleImage (format,w\_in,h\_in,type\_in,\*data\_in,w\_out,h\_out,type\_out,\*data\_out);
3. data\_in adalah gambar inputan.
4. data\_out adalah gambar hasil

## 2.10.4. Mapping Tekstur :

1. Didasarkan pada koordinat tekstur parametric
2. glTexCoord\*() ditetapkan pada masing – masing vertex

## Fog

Fog adalah suatu istilah umum yang menggambarkan bentuk yang sama dari pengaruh atmosfer, yang digunakan untuk menirukan kabut, kabut tipis, asap, atau polusi. fog sangat penting di dalam aplikasi-aplikasi simulasi yang visual, yang dibatasi oleh jarak penglihatan untuk itu perlu lebih mendekati objek. Juga sering disatukan ke dalam tampilan simulator penerbangan.

Ketika fog memungkinkan, sudut pandang objek bersifat lebih jauh untuk memudarkan warna fog. Anda dapat mengendalikan kepadatan fog, yang mana menentukan tingkat objek memudar dengan jarak meningkat, seperti juga warna fog. Fog terdapat di dalam kedua gaya yaitu gaya RGBA dan indeks warna, meski perhitungannya sedikit berbeda di kedua gaya

## Pengertian OpenGl

OpenGL(Open Graphic Library) adalah sebuah library terdiri dari berbagai macam fungsi dan biasanya digunakan untuk menggambar sebuah objek 2D ata 3D.

Selain OpenGL ada juga Graphic Library lainnya, yaitu directX (microsoft) memiliki fungsi yang hampir sama, namun DirectX lebih banyak digunakan oleh sebagian besar game developer karena beberapa fungsi DirectX nampaknya lebih memudahkan game developer untuk membuat game.

OpenGL biasanya digunakan dengan bahasa pemrograman C/C++, tapi bisa juga menggunakan java. OpenGL merupakan sebuah library untuk pemrograman grafik (Graphics Programming / GP). Dasar untuk mempelajari GP adalah terletak pada fungsi matematikanya, terutama operasi matriks. Pada GP yang perlu dipelajari adalah mengenai shading, shape, transform (rotate, translation, scala).

## Sejarah OpenGL

Tahun 1980-an, mengembangkan perangkat lunak yang dapat berfungsi dengan berbagai hardware grafis adalah tantangan nyata. Pengembang perangkat lunak antarmuka dan kebiasaan menulis driver untuk setiap perangkat keras. Ini mahal dan mengakibatkan banyak duplikasi usaha.

Pada awal 1990-an, Silicon Graphics (SGI) adalah seorang pemimpin dalam grafis 3D untuk workstation. Mereka IRIS GL API dianggap keadaan seni dan menjadi standar industri de facto, membayangi terbuka berbasis standar PHIGS. Ini karena GL IRIS dianggap lebih mudah digunakan, dan karena itu mendukung modus langsung rendering. By contrast, Sebaliknya, PHIGS dianggap sulit untuk digunakan dan ketinggalan zaman dalam hal fungsionalitas.

SGI's pesaing (termasuk Sun Microsystems, Hewlett-Packard dan IBM) juga dapat membawa ke pasar 3D hardware, didukung oleh ekstensi yang dibuat pada PHIGS standar. Hal ini pada gilirannya menyebabkan pangsa pasar SGI untuk melemahkan karena lebih banyak hardware grafis 3D pemasok memasuki pasar. Dalam upaya untuk mempengaruhi pasar, SGI memutuskan untuk mengubah IrisGL API menjadi standar terbuka.

SGI menganggap bahwa IrisGL API itu sendiri tidak cocok untuk membuka karena masalah lisensi dan paten. Juga, IrisGL memiliki fungsi-fungsi API yang tidak relevan dengan grafis 3D. Sebagai contoh, termasuk windowing, keyboard dan mouse API, sebagian karena dikembangkan sebelum Sistem X Window dan Sun's NEWS sistem dikembangkan.

Selain itu, SGI memiliki sejumlah besar pelanggan perangkat lunak; dengan mengubah ke OpenGL API mereka berencana untuk mempertahankan pelanggan mereka terkunci ke SGI (dan IBM) hardware untuk beberapa tahun sementara pasar dukungan untuk OpenGL matang. Sementara itu, SGI akan terus berusaha untuk mempertahankan pelanggan mereka terikat pada hardware SGI dengan mengembangkan maju dan kepemilikan Iris Inventor dan Iris Performer pemrograman API. Akibatnya, SGI merilis standar OpenGL.

Akses standar yang OpenGL hardware, dan mendorong tanggung jawab pengembangan program antarmuka perangkat keras, kadang-kadang disebut device driver, hardware produsen dan windowing didelegasikan fungsinya kepada sistem operasi. Dengan begitu banyak jenis perangkat keras grafis, membuat mereka semua berbicara dengan bahasa yang sama dengan cara ini mempunyai dampak luar biasa dengan memberikan pengembang perangkat lunak tingkat yang lebih tinggi 3D-platform untuk pengembangan perangkat lunak.

SGI memimpin pembentukan dewan peninjau arsitektur OpenGL (OpenGL ARB), kelompok perusahaan yang akan memelihara dan memperluas spesifikasi OpenGL selama bertahun-tahun yang akan datang. Salah satu batasan dari IrisGL adalah bahwa ia hanya memberikan akses ke fitur yang didukung oleh hardware yang mendasarinya. Jika hardware grafis tidak mendukung fitur, maka aplikasi tidak bisa menggunakannya. OpenGL mengatasi masalah ini dengan menyediakan dukungan perangkat lunak untuk fitur yang tidak didukung oleh hardware, memungkinkan aplikasi untuk menggunakan gambar pada lanjutan relatif rendah sistem bertenaga.

Tahun 1994 SGI bermain-main dengan gagasan untuk melepaskan sesuatu yang disebut "OpenGL + +" yang mencakup unsur-unsur seperti grafik adegan-API (mungkin berbasis di sekitar mereka Performer teknologi). Spesifikasi itu beredar di kalangan beberapa pihak yang berkepentingan - tetapi tidak pernah berubah menjadi sebuah produk pada tahun 1995, yang akan menjadi pesaing utama dari OpenGL. Pada tanggal 17 Desember 1997 Microsoft dan SGI memprakarsai Fahrenheit proyek, yang merupakan upaya bersama dengan tujuan mempersatukan OpenGL dan Direct3D antarmuka (dan menambahkan adegan-API grafik juga). Pada tahun 1998 Hewlett-Packard bergabung dengan proyek ini. Ini awalnya menunjukkan beberapa janji untuk membawa dunia komputer grafis 3D interaktif API, melainkan karena kendala keuangan di SGI, alasan strategis di Microsoft, dan kurangnya dukungan industri , itu ditinggalkan pada tahun 1999.

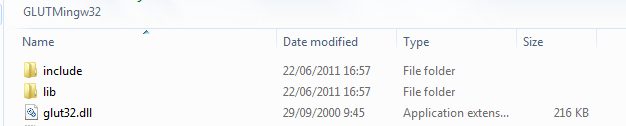
## Konfigurasi OpenGL pada Codeblocks

Setelah melakukan instalasi Codeblocks, maka langkah berikutnya yaitu konfigurasi OpenGL yang dibutuhkan komputer agar bisa membuat program grafis pada Codeblocks. File penting yang dibutuhkan untuk instalasi OpenGL adalah sebagai berikut:

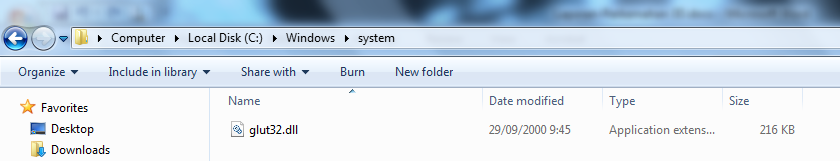
1. glut.h
2. glut32.lib
3. glut32.dll

Berikut langkah-langkah konfigurasi OpenGL pada Codeblocks:

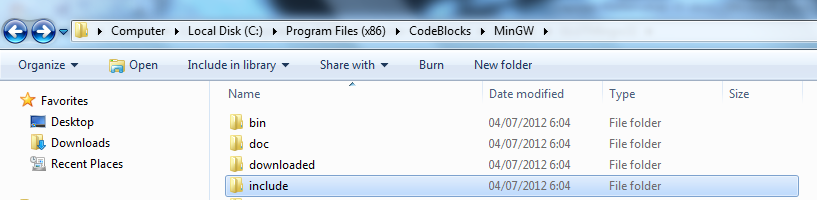
1. Ektrak file GLUTMingw32, ini adalah komponen opengl sendiri



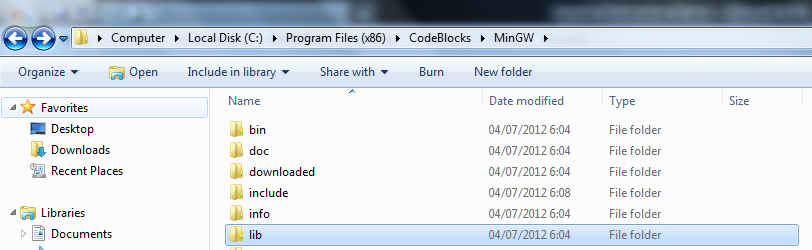
1. Copylah atau tambahkan file **glut32.dll** pada directory C:\WINDOWS\system. Seperti gambar berikut:



1. Copykan dan merge folder **include** yang ada di folder GLUTMingw32 pada directory C:\Program Files (x86)\CodeBlocks\MinGW. Seperti gambar berikut:



1. Copykan dan merge folder **lib** yang ada di folder GLUTMingw32 pada directory C:\Program Files (x86)\CodeBlocks\MinGW. Seperti gambar berikut:



1. Selesailah konfigurasi OpenGL, dan anda sudah siap membuat program grafis pada Codeblocks.

# BAB III

# IMPLEMENTASI PROGRAM

**Sourcecode Program OpenGL**

**main.cpp**

#pragma once

#ifndef \_WIN32\_WINNT // Allow use of features specific to Windows XP or later.

#define \_WIN32\_WINNT 0x0501 // Change this to the appropriate value to target other versions of Windows.

#endif

#include <stdio.h>

#include <tchar.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <windows.h>

#ifdef \_\_APPLE\_\_

#include <OpenGL/OpenGL.h>

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#include <GL/glu.h>

#include <GL/gl.h>

#include "imageloader.h"

#include "vec3f.h"

#endif

static GLfloat spin, spin2 = 0.0;

float angle = 0;

float angles = 0.0f;

float deltaAngle = 0.0f;

float deltaMove = 0;

int xOrigin = -1;

//float x=0.0f, z=15.0f;

float x=0.0f, z=15.0f;

float lx=0.0f,lz=-1.0f;

float home=15.0f,end=15.0f;

//float home=85.0f,end=15.0f;

using namespace std;

float lastx, lasty;

GLint stencilBits;

static int viewx = 50;

static int viewy = 24;

static int viewz = 80;

static float mv\_c=1, mv\_c1=1, mv=1, mv1=1, mv2=1, mv\_r=1, mv\_r1=1;

static float mva\_c=1, mva\_c1=1, mva=1, mva1=1, mva2=1, mva3=1, mva4=1, mva5=1;

float rot = 0;

// variables to compute frames per second

int frame;

long time, timebase;

char s[60];

char currentMode[80];

/\*char gameModeString[40] = "800x600";

void init();\*/

void renderBitmapString(

float x,

float y,

float z,

void \*font,

char \*string) {

char \*c;

glRasterPos3f(x, y,z);

for (c=string; \*c != '\0'; c++) {

glutBitmapCharacter(font, \*c);

}

}

//train 2D

//class untuk terain 2D

class Terrain {

private:

int w; //Width

int l; //Length

float\*\* hs; //Heights

Vec3f\*\* normals;

bool computedNormals; //Whether normals is up-to-date

public:

Terrain(int w2, int l2) {

w = w2;

l = l2;

hs = new float\*[l];

for (int i = 0; i < l; i++) {

hs[i] = new float[w];

}

normals = new Vec3f\*[l];

for (int i = 0; i < l; i++) {

normals[i] = new Vec3f[w];

}

computedNormals = false;

}

~Terrain() {

for (int i = 0; i < l; i++) {

delete[] hs[i];

}

delete[] hs;

for (int i = 0; i < l; i++) {

delete[] normals[i];

}

delete[] normals;

}

int width() {

return w;

}

int length() {

return l;

}

//Sets the height at (x, z) to y

void setHeight(int x, int z, float y) {

hs[z][x] = y;

computedNormals = false;

}

//Returns the height at (x, z)

float getHeight(int x, int z) {

return hs[z][x];

}

//Computes the normals, if they haven't been computed yet

void computeNormals() {

if (computedNormals) {

return;

}

//Compute the rough version of the normals

Vec3f\*\* normals2 = new Vec3f\*[l];

for (int i = 0; i < l; i++) {

normals2[i] = new Vec3f[w];

}

for (int z = 0; z < l; z++) {

for (int x = 0; x < w; x++) {

Vec3f sum(0.0f, 0.0f, 0.0f);

Vec3f out;

if (z > 0) {

out = Vec3f(0.0f, hs[z - 1][x] - hs[z][x], -1.0f);

}

Vec3f in;

if (z < l - 1) {

in = Vec3f(0.0f, hs[z + 1][x] - hs[z][x], 1.0f);

}

Vec3f left;

if (x > 0) {

left = Vec3f(-1.0f, hs[z][x - 1] - hs[z][x], 0.0f);

}

Vec3f right;

if (x < w - 1) {

right = Vec3f(1.0f, hs[z][x + 1] - hs[z][x], 0.0f);

}

if (x > 0 && z > 0) {

sum += out.cross(left).normalize();

}

if (x > 0 && z < l - 1) {

sum += left.cross(in).normalize();

}

if (x < w - 1 && z < l - 1) {

sum += in.cross(right).normalize();

}

if (x < w - 1 && z > 0) {

sum += right.cross(out).normalize();

}

normals2[z][x] = sum;

}

}

//Smooth out the normals

const float FALLOUT\_RATIO = 0.5f;

for (int z = 0; z < l; z++) {

for (int x = 0; x < w; x++) {

Vec3f sum = normals2[z][x];

if (x > 0) {

sum += normals2[z][x - 1] \* FALLOUT\_RATIO;

}

if (x < w - 1) {

sum += normals2[z][x + 1] \* FALLOUT\_RATIO;

}

if (z > 0) {

sum += normals2[z - 1][x] \* FALLOUT\_RATIO;

}

if (z < l - 1) {

sum += normals2[z + 1][x] \* FALLOUT\_RATIO;

}

if (sum.magnitude() == 0) {

sum = Vec3f(0.0f, 1.0f, 0.0f);

}

normals[z][x] = sum;

}

}

for (int i = 0; i < l; i++) {

delete[] normals2[i];

}

delete[] normals2;

computedNormals = true;

}

//Returns the normal at (x, z)

Vec3f getNormal(int x, int z) {

if (!computedNormals) {

computeNormals();

}

return normals[z][x];

}

};

//end class

//Loads a terrain from a heightmap. The heights of the terrain range from

//-height / 2 to height / 2.

//load terain di procedure inisialisasi

Terrain\* loadTerrain(const char\* filename, float height) {

Image\* image = loadBMP(filename);

Terrain\* t = new Terrain(image->width, image->height);

for (int y = 0; y < image->height; y++) {

for (int x = 0; x < image->width; x++) {

unsigned char color = (unsigned char) image->pixels[3 \* (y

\* image->width + x)];

float h = height \* ((color / 255.0f) - 0.4f);

t->setHeight(x, y, h);

}

}

delete image;

t->computeNormals();

return t;

}

void initRendering() {

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_NORMALIZE);

glShadeModel(GL\_SMOOTH);

}

float \_angle = 60.0f;

//buat tipe data terain

Terrain\* \_terrain;

Terrain\* \_terrainTanah;

Terrain\* \_terrainAir;

const GLfloat light\_ambient[] = { 0.3f, 0.3f, 0.3f, 1.0f };

const GLfloat light\_diffuse[] = { 0.7f, 0.7f, 0.7f, 1.0f };

const GLfloat light\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };

const GLfloat light\_position[] = { 1.0f, 1.0f, 1.0f, 1.0f };

const GLfloat light\_ambient2[] = { 0.3f, 0.3f, 0.3f, 0.0f };

const GLfloat light\_diffuse2[] = { 0.3f, 0.3f, 0.3f, 0.0f };

const GLfloat mat\_ambient[] = { 0.8f, 0.8f, 0.8f, 1.0f };

const GLfloat mat\_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };

const GLfloat mat\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };

const GLfloat high\_shininess[] = { 100.0f };

void cleanup() {

delete \_terrain;

delete \_terrainTanah;

delete \_terrainAir;

}

//untuk di display

void drawSceneTanah(Terrain \*terrain, GLfloat r, GLfloat g, GLfloat b) {

float scale = 1000.0f / max(terrain->width() - 1, terrain->length() - 1);

glScalef(scale, scale, scale);

glTranslatef(-(float) (terrain->width() - 1) / 2, 0.0f,

-(float) (terrain->length() - 1) / 2);

glColor3f(r, g, b);

for (int z = 0; z < terrain->length() - 1; z++) {

//Makes OpenGL draw a triangle at every three consecutive vertices

glBegin(GL\_TRIANGLE\_STRIP);

for (int x = 0; x < terrain->width(); x++) {

Vec3f normal = terrain->getNormal(x, z);

glNormal3f(normal[0], normal[1], normal[2]);

glVertex3f(x, terrain->getHeight(x, z), z);

normal = terrain->getNormal(x, z + 1);

glNormal3f(normal[0], normal[1], normal[2]);

glVertex3f(x, terrain->getHeight(x, z + 1), z + 1);

}

glEnd();

}

}

//=================================================================

// House

//=================================================================

void jendela()

{

//window

glPushMatrix();

glTranslatef(-18.0f, 5.0f, 15.0f);

glColor4f(0.35, 0.15, 0.0, 0.0);

glScalef(1.5,1,0.1);

glutSolidCube(7.0f);

glPopMatrix();

//kaca

glPushMatrix();

glTranslatef(-18.0f, 5.0f, 15.2f);

glColor3f(0.5, 1, 1);

glScalef(2.4,1.5,0.1);

glutSolidCube(4.0f);

glPopMatrix();

}

void drawRumah\_hijau()

{

//Wall

glPushMatrix();

glTranslatef(0.0f, 0.0f, -5.0f);

glColor3f(0.5, 1, 0.5);

glScalef(2.5,1,1);

glutSolidCube(20.0f);

glPopMatrix();

glPushMatrix();

glTranslatef(-13.0f, 0.0f, 10.0f);

glColor3f(0.5, 1, 0.5);

glScalef(1.2,1,0.5);

glutSolidCube(20.0f);

glPopMatrix();

//cerobong asap

glPushMatrix();

glTranslatef(-15.0f, 10.1f, -5.0f);

glColor3f(0, 0, 0);

glScalef(0.2,1.0,0.2);

glutSolidCube(20.0f);

glPopMatrix();

//Pintu

glPushMatrix();

glTranslatef(-5.0f, 2.0f, 15.0f);

glColor4f(0.35, 0.15, 0.2, 1.0);

glScalef(0.6,1.3,0.1);

glutSolidCube(10.0f);

glPopMatrix();

glPushMatrix();

glTranslatef(-7.0f, 1.0f, 15.8f);

glColor4f(0.35, 0.15, 0.2, 1.0);

glutSolidSphere(0.5f,50.0f,25.0f);

glPopMatrix();

//window

jendela();

glPushMatrix();

glTranslatef(28.0f, 1.0f, 3.55f);

glScalef(0.4,0.85,0.1);

jendela();

glPopMatrix();

glPushMatrix();

glTranslatef(22.0f, 1.0f, 3.55f);

glScalef(0.4,0.85,0.1);

jendela();

glPopMatrix();

glPushMatrix();

glTranslatef(16.0f, 1.0f, 3.55f);

glScalef(0.4,0.85,0.1);

jendela();

glPopMatrix();

return;

}

//=================================================================

// Matras

//=================================================================

void drawMatras()

{

glColor3f(0.9, 0.9, 0.9);

glPushMatrix();

glTranslatef(-1.2, 0.0, -0.5);

glScalef(1.2, 0.05, 2);

glRotatef(90, 1, 0, 0);

glutSolidCube(7.0f);

glPopMatrix();

return;

}

//=================================================================

// Drone Toy

//=================================================================

void drawFan()

{

glPushMatrix();

glTranslatef(0.0f, 4.0f, -8.0f);

glRotatef(90.0f, 0.0f, 1.0f, 0.0f);

glRotatef(0.0f, 1.0f, 0.0f, 0.0f);

glColor3f(1.0f, 0.0f, 0.0f);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

//kiri

int i = 1;

GLfloat xS = 20;

GLfloat yS = 12;

GLfloat zS = 50;

//for (int i = 1; i <= 3; i++) {

glPushMatrix();

glTranslatef(0.0f, 4.0f, -8.0f);

//glTranslatef(xS - 6, yS - 5 - 1, zS - 8.2);

glRotatef(90\*i, 1\*i, 0, 0);

//glRotatef(-15 + (i\*8), 0, 1, 0);

//glRotatef(10.5, 0, 0, 1);

glScalef(4\*4.5-i, 0.6 \* 3, 2.5 \* 3 - 1);

glNormal3f(-0.97014254f, 0.0f, 0.24253564f);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

//}

//========================

glColor3f(1.0f, 1.0f, 0.0f);

glPushMatrix();

glTranslatef(0.0f, 4.0f, -8.0f);

//glTranslatef(xS - 6, yS - 5 - 1, zS - 8.2);

glRotatef(100\*i, 1, 90, 0);

glScalef(4\*4.5-i, 0.6 \* 3, 2.5 \* 3 - 1);

glNormal3f(-0.97014254f, 0.0f, 0.24253564f);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

for (int i = 0; i < 3; i++) {

//kiri

glPushMatrix();

glTranslatef(xS - 8, yS - 7 - i, zS - 9);

glRotatef(80 + 15, 1, 0, 0);

glRotatef(-15 + (i \* 38), 0, 1, 0);

//glRotatef(10.5, 0, 0, 1);

glScalef(4 \* 4.5 - i, 0.6 \* 3, 2.5 \* 3 - 1);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

}

return;

}

//=================================================================

// Duck Toy

//=================================================================

void drawBebek() {

GLfloat xS = 20;

GLfloat yS = 12;

GLfloat zS = 50;

//kepala

glColor3f(0.88, 0.88, 0.1);

//glPushMatrix();

glPushMatrix();

glTranslatef(xS + 1, yS + 3, zS);

glScalef(3 \* 4, 3 \* 4, 3.5 \* 4);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

glColor3f(0.8, 0.3, 0.4);

//glPushMatrix();

glPushMatrix();

glTranslatef(xS + 1, yS + 6, zS);

glScalef(3 \* 4, 3 \* 4, 0.5 \* 4);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

GLfloat x = 0.6 - 50;

GLfloat y = 2 + 12;

GLfloat z = 1.4 + 24;

glColor3f(0.7, 0.6, 0.4);

glPushMatrix();

glRotatef(95, 0, 1, 0);

glTranslatef(x - 4.4, y + 2, z - 4.6);

glScalef(1.5 \* 4, 1.5 \* 4, 1.5 \* 4);

glutSolidSphere(0.3, 20, 30);

glPopMatrix();

glColor3f(0.0, 0.0, 0.0);

glPushMatrix();

glRotatef(95, 0, 1, 0);

glTranslatef(x - 4.7, y + 2.1, z - 3.4);

glScalef(1.5 \* 3, 1.5 \* 3, 1.5 \* 3);

glutSolidSphere(0.18, 20, 30);

glPopMatrix();

glPushMatrix();

glColor3f(0.7, 0.6, 0.4);

glRotatef(83, 0, 1, 0);

glTranslatef(x + 4.9, y + 2.1, z + 5.8);

glScalef(1.5 \* 4, 1.5 \* 4, 1.5 \* 4);

glutSolidSphere(0.3, 20, 30);

glPopMatrix();

glPushMatrix();

glColor3f(0.0, 0.0, 0.0);

glRotatef(83, 0, 1, 0);

glTranslatef(x + 4.9, y + 2.1, z + 6.65);

glScalef(1.5 \* 4, 1.5 \* 4, 1.5 \* 4);

glutSolidSphere(0.18, 20, 30);

glPopMatrix();

glColor3f(0.88, 0.88, 0.3);

//mulut

glPushMatrix();

glTranslatef(xS + 8, yS + 2, zS);

glRotatef(180, 0, 0, 1);

glScalef(4 \* 3, 0.6 \* 3, 2.5 \* 3);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

//

glColor3f(0.88, 0.2, 0.5);

glPushMatrix();

glTranslatef(xS + 1, yS - 3, zS);

glScalef(3 \* 4, 3 \* 4, 3.5 \* 4);

glRotatef(90, 1, 0, 0);

glutSolidTorus(0.2, 0.3, 20, 20);

//badan agak atas

glPopMatrix();

glColor3f(0.88, 0.88, 0.1);

glPushMatrix();

glTranslatef(xS + 1, yS - 6, zS - 0.4);

glScalef(3 \* 4, 3 \* 4, 3.5 \* 4);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

//badan utama

glPushMatrix();

glColor3f(0.88, 0.88, 0.1);

glTranslatef(xS - 7, yS - 10, zS - 0.3);

glScalef(5 \* 4, 4 \* 4, 4 \* 4);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

//badan depan bawah

glPushMatrix();

glColor3f(0.88, 0.88, 0.1);

glTranslatef(xS, yS - 8, zS - 0.2);

glScalef(3 \* 4, 3 \* 4, 3.5 \* 4);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

//

int sudutSayap = 80;

//sayap

for (int i = 0; i < 3; i++) {

//kanan

glPushMatrix();

glTranslatef(xS - 6, yS - 5 - i, zS + 7);

glRotatef(sudutSayap, 1, 0, 0);

glRotatef(-15 + (i \* 8), 0, 1, 0);

glRotatef(-8.5, 0, 0, 1);

glScalef(4 \* 4.5 - i, 0.6 \* 3, 2.5 \* 3 - 1);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

//kiri

glPushMatrix();

glTranslatef(xS - 6, yS - 5 - i, zS - 8.2);

glRotatef(sudutSayap + 15, 1, 0, 0);

glRotatef(-15 + (i \* 8), 0, 1, 0);

glRotatef(10.5, 0, 0, 1);

glScalef(4 \* 4.5 - i, 0.6 \* 3, 2.5 \* 3 - 1);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

}

//buntut

int sudut = 145;

for (int i = 0; i < 4; i++) {

glPushMatrix();

glTranslatef(xS - 12.5 + i, yS - 5 - i, zS);

glRotatef(sudut, 0, 0, 1);

glScalef(4 \* 6.5, 0.6 \* 6, (2.5 \* 4) - i);

glutSolidSphere(0.5, 20, 30);

glPopMatrix();

sudut += 10;

}

glPopMatrix();

}

unsigned int LoadTextureFromBmpFile(char \*filename);

//Meja

void Meja()

{

// Draw kubus

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kaki kanan depan

glTranslatef(1.0f, -2.0f, 3.0f);

glScaled(1.0, 3.0, 1.0);

glutSolidCube(0.5f);

glPopMatrix();

// Draw kubus

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kaki kanan belakang

glTranslatef(1.0f, -2.0f, 1.0f);

glScaled(1.0, 3.0, 1.0);

glutSolidCube(0.5f);

glPopMatrix();

// Draw kubus

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kaki kiri belakang

glTranslatef(-3.0f, -2.0f, 1.0f);

glScaled(1.0, 3.0, 1.0);

glutSolidCube(0.5f);

glPopMatrix();

// Draw kubus

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kaki kiri depan

glTranslatef(-3.0f, -2.0f, 3.0f);

glScaled(1.0, 3.0, 1.0);

glutSolidCube(0.5f);

glPopMatrix();

// Draw kubus

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // alas meja

glTranslatef(-1.0f, -1.25f, 2.0f);

glScaled(9.0, 0.5 , 5.0);

glutSolidCube(0.5f);

glPopMatrix();

}

//Kayu Bakar

void KayuBakar()

{

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu kanan bawah

glTranslatef(1.1f, -3.0f, -3.0f);

glScaled(0.5, 0.5 , 5.0);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu kiri bawah

glTranslatef(-1.1f, -3.0f, -3.0f);

glScaled(0.5, 0.5 , 5.0);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu depan bawah

glTranslatef(0.0f, -2.75f, -1.85f);

glScaled(5.0, 0.5 , 0.5);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu belakang bawah

glTranslatef(0.0f, -2.75f, -4.1f);

glScaled(5.0, 0.5 , 0.5);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu kanan bawah 1

glTranslatef(1.1f, -2.5f, -3.0f);

glScaled(0.5, 0.5 , 5.0);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu kiri bawah 1

glTranslatef(-1.1f, -2.5f, -3.0f);

glScaled(0.5, 0.5 , 5.0);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu depan bawah 1

glTranslatef(0.0f, -2.25f, -1.85f);

glScaled(5.0, 0.5 , 0.5);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu belakang bawah 1

glTranslatef(0.0f, -2.25f, -4.1f);

glScaled(5.0, 0.5 , 0.5);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu kanan bawah 2

glTranslatef(1.1f, -2.0f, -3.0f);

glScaled(0.5, 0.5 , 5.0);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu kiri bawah 2

glTranslatef(-1.1f, -2.0f, -3.0f);

glScaled(0.5, 0.5 , 5.0);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu depan bawah 2

glTranslatef(0.0f, -1.75f, -1.85f);

glScaled(5.0, 0.5 , 0.5);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu belakang bawah 2

glTranslatef(0.0f, -1.75f, -4.1f);

glScaled(5.0, 0.5 , 0.5);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu kanan bawah 3

glTranslatef(1.1f, -1.5f, -3.0f);

glScaled(0.5, 0.5 , 5.0);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu kiri bawah 3

glTranslatef(-1.1f, -1.5f, -3.0f);

glScaled(0.5, 0.5 , 5.0);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu depan bawah 3

glTranslatef(0.0f, -1.25f, -1.85f);

glScaled(5.0, 0.5 , 0.5);

glutSolidCube(0.5f);

glPopMatrix();

//kayu

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kayu belakang bawah 3

glTranslatef(0.0f, -1.25f, -4.1f);

glScaled(5.0, 0.5 , 0.5);

glutSolidCube(0.5f);

glPopMatrix();

}

void Bintang()

{

glPushMatrix();

glColor4f(1.0f, 1.0f, 1.0f, 1.0f);

glTranslatef(0.0f, -1.25f, -4.1f);

glScaled(0.5, 0.5, 0.5);

glutSolidSphere(1.0f, 100, 10);

glPopMatrix();

}

void Kursi()

{

// Draw kubus

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kanan depan

glTranslatef(-1.0f, -2.0f, 3.0f);

glScaled(1.0, 3.0, 1.0);

glutSolidCube(0.5f);

glPopMatrix();

// Draw kubus

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kanan belakang

glTranslatef(-1.0f, -1.0f, 1.0f);

glScaled(1.0, 7.0, 1.0);

glutSolidCube(0.5f);

glPopMatrix();

// Draw kubus

glPushMatrix();

glColor4f(0.35f, 0.15f, 0.0f, 1.0f); // kiri belakang

glTranslatef(-3.0f, -1.0f, 1.0f);

glScaled(1.0, 7.0, 1.0);

glutSolidCube(0.5f);

glPopMatrix();

// Draw kubus

glPushMatrix();

glColor4f(0.35f,0.15f,0.0f,1.0f); // kiri depan

glTranslatef(-3.0f, -2.0f, 3.0f);

glScaled(1.0, 3.0, 1.0);

glutSolidCube(0.5f);

glPopMatrix();

// Draw kubus

glPushMatrix();

glColor4f(0.35f,0.15f,0.0f,1.0f); // alas

glTranslatef(-2.0f, -1.25f, 2.0f);

glScaled(5.0, 0.5 , 5.0);

glutSolidCube(0.5f);

glPopMatrix();

// Draw kubus

glPushMatrix();

glColor4f(0.35f,0.15f,0.0f,1.0f); // alas atas

glTranslatef(-2.0f, 0.24f, 1.25f);

glScaled(5.0, 2.0 , 0.25);

glutSolidCube(0.5f);

glPopMatrix();

}

void Tenda()

{

// Draw Pyramid

glPushMatrix();

glColor4f(0.0f, 0.1f, 0.0f, 0.0f);

glBegin(GL\_TRIANGLES);

glNormal3f(1,0,1);

// first triangle

glVertex3f(1,0,1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(-1,0,1); //

glVertex3f(-1,0,1); //

glNormal3f(-1,0,1);

// second triangle

glVertex3f(-1,0,1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(-1,0,-1); //

glVertex3f(-1,0,-1); //

glNormal3f(-1,0,-1);

// third triangle

glVertex3f(-1,0,-1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(1,0,-1); //

glVertex3f(1,0,-1); //

glNormal3f(1,0,-1);

// last triangle

glVertex3f(1,0,-1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(1,0,1); //

glVertex3f(1,0,1); //

glEnd();

// alas piramid

glColor4f(0.0f, 0.1f, 0.0f, 0.0f);

glBegin(GL\_TRIANGLES);

glNormal3f(-1,0,1);

glVertex3f(-1,0,1);

glNormal3f(-1,0,-1);

glVertex3f(-1,0,-1);

glNormal3f(1,0,1);

glVertex3f(1,0,1);

glNormal3f(1,0,1);

glVertex3f(1,0,1);

glNormal3f(-1,0,-1);

glVertex3f(-1,0,-1);

glNormal3f(1,0,-1);

glVertex3f(1,0,-1);

glEnd();

glPopMatrix();

}

void PintuTenda()

{

// Draw PintuTenda

glPushMatrix();

glColor4f(0.3f, 0.0f, 0.0f, 0.0f);

glBegin(GL\_TRIANGLES);

glNormal3f(1,0,1);

// first triangle

glVertex3f(1,0,1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(-1,0,1); //

glVertex3f(-1,0,1); //

glNormal3f(-1,0,1);

// second triangle

glVertex3f(-1,0,1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(-1,0,-1); //

glVertex3f(-1,0,-1); //

glNormal3f(-1,0,-1);

// third triangle

glVertex3f(-1,0,-1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(1,0,-1); //

glVertex3f(1,0,-1); //

glNormal3f(1,0,-1);

// last triangle

glVertex3f(1,0,-1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(1,0,1); //

glVertex3f(1,0,1); //

glEnd();

// alas piramid

glColor4f(0.0f, 0.1f, 0.0f, 0.0f);

glBegin(GL\_TRIANGLES);

glNormal3f(-1,0,1);

glVertex3f(-1,0,1);

glNormal3f(-1,0,-1);

glVertex3f(-1,0,-1);

glNormal3f(1,0,1);

glVertex3f(1,0,1);

glNormal3f(1,0,1);

glVertex3f(1,0,1);

glNormal3f(-1,0,-1);

glVertex3f(-1,0,-1);

glNormal3f(1,0,-1);

glVertex3f(1,0,-1);

glEnd();

glPopMatrix();

}

void AtapRumah()

{

// Draw atap

glPushMatrix();

glColor4f(0.0f, 0.1f, 0.5f, 0.0f);

glBegin(GL\_TRIANGLES);

glNormal3f(1,0,1);

// first triangle

glVertex3f(1,0,1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(-1,0,1); //

glVertex3f(-1,0,1); //

glNormal3f(-1,0,1);

// second triangle

glVertex3f(-1,0,1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(-1,0,-1); //

glVertex3f(-1,0,-1); //

glNormal3f(-1,0,-1);

// third triangle

glVertex3f(-1,0,-1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(1,0,-1); //

glVertex3f(1,0,-1); //

glNormal3f(1,0,-1);

// last triangle

glVertex3f(1,0,-1); //

glNormal3f(0,2,0); //

glVertex3f(0,2,0); //

glNormal3f(1,0,1); //

glVertex3f(1,0,1); //

glEnd();

// alas piramid

glColor4f(0.0f, 0.1f, 0.5f, 0.0f);

glBegin(GL\_TRIANGLES);

glNormal3f(-1,0,1);

glVertex3f(-1,0,1);

glNormal3f(-1,0,-1);

glVertex3f(-1,0,-1);

glNormal3f(1,0,1);

glVertex3f(1,0,1);

glNormal3f(1,0,1);

glVertex3f(1,0,1);

glNormal3f(-1,0,-1);

glVertex3f(-1,0,-1);

glNormal3f(1,0,-1);

glVertex3f(1,0,-1);

glEnd();

glPopMatrix();

}

void pohon1()

{

// Draw daun bulet"

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 5.24f, 0.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

// tengah

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 4.14f, 1.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 4.8f, 0.8f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 3.4f, 1.1f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 2.7f, 0.7f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

// daun kanan

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(1.0f, 4.8f, 0.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(1.1f, 4.15f, 0.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(1.0f, 3.4f, 0.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.7f, 2.7f, 0.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

// daun kiri

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-1.0f, 4.8f, 0.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-1.1f, 4.15f, 0.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-1.0f, 3.4f, 0.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-0.7f, 2.7f, 0.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

// daun belakang

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 4.14f, -1.0f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 4.8f, -0.8f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 3.4f, -1.1f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 2.7f, -0.7f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

// serong kanan

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.7f, 4.8f, 0.7f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(1.0f, 4.15f, 0.8f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.8f, 3.4f, 0.8f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.7f, 2.7f, 0.7f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

// serong kiri

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-0.7f, 4.8f, 0.7f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-1.0f, 4.15f, 0.8f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-0.8f, 3.4f, 0.8f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-0.7f, 2.7f, 0.7f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

// serong kanan belakang

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.7f, 4.8f, -0.7f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(1.0f, 4.15f, -0.8f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.8f, 3.4f, -0.8f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.7f, 2.7f, -0.7f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

// serong kiri belakang

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-0.7f, 4.8f, -0.7f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-1.0f, 4.15f, -0.8f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-0.8f, 3.4f, -0.8f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(-0.7f, 2.7f, -0.7f);

glScaled(3,3,3);

glutSolidSphere(0.25f,50,50);

glPopMatrix();

// Draw batang

glPushMatrix();

glColor3f(0.8f, 0.4f, 0.1f); // coklat

glTranslatef(0.0f, 1.5f, 0.0f);

glScaled(1.1,6,1);

glutSolidCube(0.5f);

glPopMatrix();

}

void pohon2()

{

// Draw daun cone

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 2, 0.0f);

glScaled(2.5,2.5,2.5);

glRotated(-90,1,0,0);

glutSolidCone(1,2,60,1);

glPopMatrix();

// Draw batang

glPushMatrix();

glColor3f(0.8f, 0.4f, 0.1f); // coklat

glTranslatef(0.0f, 1.5f, 0.0f);

glScaled(1.1,6,1);

glutSolidCube(0.5f);

glPopMatrix();

}

void pohon3()

{

// Draw daun octahedron

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 4.6f, 0.0f);

glScaled(2.5,2.5,2.5);

glutSolidOctahedron();

glPopMatrix();

// Draw batang

glPushMatrix();

glColor3f(0.8f, 0.4f, 0.1f); // coklat

glTranslatef(0.0f, 1.5f, 0.0f);

glScaled(1.1,6,1);

glutSolidCube(0.5f);

glPopMatrix();

}

void pohon4()

{

// Draw daun bulet

glPushMatrix();

glColor3f(0.0f, 0.3f, 0.0f);

glTranslatef(0.0f, 3.24f, 0.0f);

glScaled(7,7,7);

glutSolidSphere(0.25f,5,50);

glPopMatrix();

// Draw batang

glPushMatrix();

glColor3f(0.8f, 0.4f, 0.1f); // coklat

glTranslatef(0.0f, 1.5f, 0.0f);

glScaled(1.1,6,1);

glutSolidCube(0.5f);

glPopMatrix();

}

void drawBarang()

{

//teko

glPushMatrix();

glTranslatef(63, 0, -240);

glScalef(0.2, 0.2, 0.2);

glPopMatrix();

//buku

glPushMatrix();

drawMatras();

glPopMatrix();

}

void computePos(float deltaMove) {

x += deltaMove \* lx \* 0.25f;

z += deltaMove \* lz \* 0.25f;

}

void display(void) {

if (deltaMove)

computePos(deltaMove);

glClearStencil(0); //clear the stencil buffer

glClearDepth(1.0f);

//glClearColor(0.0, 0.6, 0.8, 1); //warna langit

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT); //clear the buffers

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt( x, home, z,

x+lx, 15.0f, z+lz,

0.0f, 1.0f, 0.0f);

glPushMatrix();

drawSceneTanah(\_terrain, 0.4f, 1.0f, 0.1f);

glPopMatrix();

glPushMatrix();

drawSceneTanah(\_terrainTanah, 0.6f, 0.2f, 0.1f);

glPopMatrix();

glPushMatrix();

drawSceneTanah(\_terrainAir, 0.1f, 0.2f, 0.9f);

glPopMatrix();

//Rumah

glPushMatrix();

glTranslatef(240.0,0.0,-180.0);

drawRumah\_hijau();

glPopMatrix();

glPushMatrix();

glTranslatef(180.0,0.0,-180.0);

drawRumah\_hijau();

glPopMatrix();

glPushMatrix();

glTranslatef(120.0,0.0,-180.0);

drawRumah\_hijau();

glPopMatrix();

glPushMatrix();

glTranslatef(60.0,0.0,-180.0);

drawRumah\_hijau();

glPopMatrix();

//barang;

for(int i=1; i<=4; i++)

{

glPushMatrix();

glTranslatef(3+i\*60, 0.5, -225);

glScalef(0.2, 0.2, 0.2);

drawBarang();

glPopMatrix();

}

//matras

glPushMatrix();

glTranslatef(33, -4.5, -220);

glScalef(2, 2, 2);

drawMatras();

glPopMatrix();

//fan

//drawFan();

//sample bebek1

glPushMatrix();

glTranslatef(400-(1.5\*mv)+(1.5\*mv1), -11, 80+(mv\*0.35)-(mv1\*0.35));

glScalef(0.2, 0.2, 0.2);

glRotatef(195, 0, 2, 0);

drawBebek();

glPopMatrix();

//sample bebek2

glPushMatrix();

glTranslatef(140+(1.5\*mv)-(1.5\*mv1), -11, 95-(mv\*0.35)+(mv1\*0.35));

//glTranslatef(400-(1.5\*mv2), -10, 80+(mv2\*0.35));

glScalef(0.3, 0.3, 0.3);

glRotatef(195, 0, 2, 0);

glRotated(175, 0, 1, 0);

if(mv\_c==0)

{glRotated((mv\_r-mv\_r1), 0, 1, 0);}

drawBebek();

glPopMatrix();

//sample bebek2\_a

glPushMatrix();

glTranslatef(135+(1.5\*mv)-(1.5\*mv1), -11, 95-(mv\*0.35)+(mv1\*0.35));

//glTranslatef(400-(1.5\*mv2), -10, 80+(mv2\*0.35));

glScalef(0.1, 0.1, 0.1);

glRotatef(195, 0, 2, 0);

glRotated(175, 0, 1, 0);

if(mv\_c==0)

{glRotated((mv\_r-mv\_r1), 0, 1, 0);}

drawBebek();

glPopMatrix();

//sample bebek2\_b

glPushMatrix();

glTranslatef(128+(1.5\*mv)-(1.5\*mv1), -11, 90-(mv\*0.35)+(mv1\*0.35));

//glTranslatef(400-(1.5\*mv2), -10, 80+(mv2\*0.35));

glScalef(0.1, 0.1, 0.1);

glRotatef(195, 0, 2, 0);

glRotated(175, 0, 1, 0);

if(mv\_c==0)

{glRotated((mv\_r-mv\_r1), 0, 1, 0);}

drawBebek();

glPopMatrix();

//sample bebek2\_c

glPushMatrix();

glTranslatef(160+(1.5\*mv)-(1.5\*mv1), -11, 95-(mv\*0.35)+(mv1\*0.35));

//glTranslatef(400-(1.5\*mv2), -10, 80+(mv2\*0.35));

glScalef(0.1, 0.1, 0.1);

glRotatef(195, 0, 2, 0);

glRotated(175, 0, 1, 0);

if(mv\_c==0)

{glRotated((mv\_r-mv\_r1), 0, 1, 0);}

drawBebek();

glPopMatrix();

//sample bebek2\_d

glPushMatrix();

glTranslatef(160+(1.5\*mv)-(1.5\*mv1), -11, 90-(mv\*0.35)+(mv1\*0.35));

//glTranslatef(400-(1.5\*mv2), -10, 80+(mv2\*0.35));

glScalef(0.1, 0.1, 0.1);

glRotatef(195, 0, 2, 0);

glRotated(175, 0, 1, 0);

if(mv\_c==0)

{glRotated((mv\_r-mv\_r1), 0, 1, 0);}

drawBebek();

glPopMatrix();

//sample bebek3

glPushMatrix();

glTranslatef(130+(0.05\*mva)-(0.5\*mva1)+mva2, -11, 120+(mva\*1.35)-(mva1\*1.35)-(mva2\*1.90));

//glTranslatef(400-(1.5\*mv2), -10, 80+(mv2\*0.35));

glScalef(0.2, 0.2, 0.2);

glRotatef(195, 0, 2, 0);

glRotated(75, 0, 1, 0);

drawBebek();

glPopMatrix();

//sample bebek4

glPushMatrix();

glTranslatef(350-(1.5\*mv)+(1.5\*mv1), -11, 80+(mv\*0.35)-(mv1\*0.35));

glScalef(0.2, 0.2, 0.2);

glRotatef(195, 0, 2, 0);

drawBebek();

glPopMatrix();

//meja 1

glPushMatrix();

glTranslatef(65, 3, -230);

glScalef(2.5f, 2.5f, 2.5f);

Meja();

glPopMatrix();

//meja 2

glPushMatrix();

glTranslatef(125, 3, -230);

glScalef(2.5f, 2.5f, 2.5f);

Meja();

glPopMatrix();

//meja 3

glPushMatrix();

glTranslatef(185, 3, -230);

glScalef(2.5f, 2.5f, 2.5f);

Meja();

glPopMatrix();

//meja 4

glPushMatrix();

glTranslatef(245, 3, -230);

glScalef(2.5f, 2.5f, 2.5f);

Meja();

glPopMatrix();

//kayubakar 1

glPushMatrix();

glTranslatef(63, 0, -240);

glScalef(1.0f, 1.0f, 1.0f);

KayuBakar();

glPopMatrix();

//kayubakar 2

glPushMatrix();

glTranslatef(123, 0, -240);

glScalef(1.0f, 1.0f, 1.0f);

KayuBakar();

glPopMatrix();

//kayubakar 3

glPushMatrix();

glTranslatef(183, 0, -240);

glScalef(1.0f, 1.0f, 1.0f);

KayuBakar();

glPopMatrix();

//kayubakar 4

glPushMatrix();

glTranslatef(243, 0, -240);

glScalef(1.0f, 1.0f, 1.0f);

KayuBakar();

glPopMatrix();

for (int i = -10; i < 10; i++)

for (int j = -10; j < 10; j++)

{

glPushMatrix();

glTranslatef(i\*280, 200, j\*200);

glScalef(3.5f, 3.5f, 3.5f);

Bintang();

glPopMatrix();

}

//kursi1

glPushMatrix();

glTranslatef(65, 0, -235);

glScalef(1.5f, 1.5f, 1.5f);

Kursi();

glPopMatrix();

//kursi2

glPushMatrix();

glTranslatef(125, 0, -235);

glScalef(1.5f, 1.5f, 1.5f);

Kursi();

glPopMatrix();

//kursi3

glPushMatrix();

glTranslatef(185, 0, -235);

glScalef(1.5f, 1.5f, 1.5f);

Kursi();

glPopMatrix();

//kursi4

glPushMatrix();

glTranslatef(245, 0, -235);

glScalef(1.5f, 1.5f, 1.5f);

Kursi();

glPopMatrix();

//kursi5

glPushMatrix();

glTranslatef(50, 0, -228);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(90.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi6

glPushMatrix();

glTranslatef(110, 0, -228);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(90.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi7

glPushMatrix();

glTranslatef(170, 0, -228);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(90.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi8

glPushMatrix();

glTranslatef(230, 0, -228);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(90.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi9

glPushMatrix();

glTranslatef(75, 0, -222);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(-90.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi10

glPushMatrix();

glTranslatef(135, 0, -222);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(-90.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi11

glPushMatrix();

glTranslatef(195, 0, -222);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(-90.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi12

glPushMatrix();

glTranslatef(255, 0, -222);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(-90.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi13

glPushMatrix();

glTranslatef(59, 0, -215);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(180.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi14

glPushMatrix();

glTranslatef(119, 0, -215);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(180.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi15

glPushMatrix();

glTranslatef(179, 0, -215);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(180.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//kursi16

glPushMatrix();

glTranslatef(239, 0, -215);

glScalef(1.5f, 1.5f, 1.5f);

glRotatef(180.0f, 0.0f , 1.0f ,0.0f);

Kursi();

glPopMatrix();

//======Tenda======

//tenda

glPushMatrix();

glTranslatef(50, -3, -250);

glScalef(7.0f, 6.0f, 7.0f);

Tenda();

glPopMatrix();

glPushMatrix();

glTranslatef(50, -3, -249.5);

glScalef(1.0f, 5.5f, 6.7f);

PintuTenda();

glPopMatrix();

//tenda2

glPushMatrix();

glTranslatef(110, -3, -250);

glScalef(7.0f, 6.0f, 7.0f);

Tenda();

glPopMatrix();

glPushMatrix();

glTranslatef(110, -3, -249.5);

glScalef(1.0f, 5.5f, 6.7f);

PintuTenda();

glPopMatrix();

//tenda3

glPushMatrix();

glTranslatef(170, -3, -250);

glScalef(7.0f, 6.0f, 7.0f);

Tenda();

glPopMatrix();

glPushMatrix();

glTranslatef(170, -3, -249.5);

glScalef(1.0f, 5.5f, 6.7f);

PintuTenda();

glPopMatrix();

//tenda2

glPushMatrix();

glTranslatef(230, -3, -250);

glScalef(7.0f, 6.0f, 7.0f);

Tenda();

glPopMatrix();

glPushMatrix();

glTranslatef(230, -3, -249.5);

glScalef(1.0f, 5.5f, 6.7f);

PintuTenda();

glPopMatrix();

//======Atap Rumah======

//atap1

glPushMatrix();

glTranslatef(73, 10, -185);

glScalef(14.0f, 5.0f, 11.0f);

AtapRumah();

glPopMatrix();

glPushMatrix();

glTranslatef(47, 10, -170);

glScalef(13.0f, 4.0f, 5.0f);

AtapRumah();

glPopMatrix();

//atap2

glPushMatrix();

glTranslatef(133, 10, -185);

glScalef(14.0f, 5.0f, 11.0f);

AtapRumah();

glPopMatrix();

glPushMatrix();

glTranslatef(107, 10, -170);

glScalef(13.0f, 4.0f, 5.0f);

AtapRumah();

glPopMatrix();

//atap3

glPushMatrix();

glTranslatef(193, 10, -185);

glScalef(14.0f, 5.0f, 11.0f);

AtapRumah();

glPopMatrix();

glPushMatrix();

glTranslatef(167, 10, -170);

glScalef(13.0f, 4.0f, 5.0f);

AtapRumah();

glPopMatrix();

//atap4

glPushMatrix();

glTranslatef(253, 10, -185);

glScalef(14.0f, 5.0f, 11.0f);

AtapRumah();

glPopMatrix();

glPushMatrix();

glTranslatef(227, 10, -170);

glScalef(13.0f, 4.0f, 5.0f);

AtapRumah();

glPopMatrix();

//======pohon======

//pohon1

glPushMatrix();

glTranslatef(70,-2.5,-270);

glScalef(7,7,7);

pohon1();

glPopMatrix();

//pohon1.1

glPushMatrix();

glTranslatef(130,-2.5,-270);

glScalef(7,7,7);

pohon1();

glPopMatrix();

//pohon1.2

glPushMatrix();

glTranslatef(190,-2.5,-270);

glScalef(7,7,7);

pohon1();

glPopMatrix();

//pohon1.3

glPushMatrix();

glTranslatef(250,-2.5,-270);

glScalef(7,7,7);

pohon1();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(73,-2.5,-150);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3.1

glPushMatrix();

glTranslatef(133,-2.5,-150);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3.2

glPushMatrix();

glTranslatef(193,-2.5,-150);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3.3

glPushMatrix();

glTranslatef(250,-2.5,-150);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon2.21

glPushMatrix();

glTranslatef(450,5,200);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.22

glPushMatrix();

glTranslatef(400,5,200);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.24

glPushMatrix();

glTranslatef(300,5,200);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.23

glPushMatrix();

glTranslatef(350,5,200);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.25

glPushMatrix();

glTranslatef(250,5,200);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.26

glPushMatrix();

glTranslatef(200,5,200);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.11

glPushMatrix();

glTranslatef(400,5,150);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.11

glPushMatrix();

glTranslatef(350,5,150);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.11

glPushMatrix();

glTranslatef(300,5,150);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.31

glPushMatrix();

glTranslatef(450,3,250);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.32

glPushMatrix();

glTranslatef(400,5,250);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.33

glPushMatrix();

glTranslatef(350,3,250);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.34

glPushMatrix();

glTranslatef(300,5,250);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.35

glPushMatrix();

glTranslatef(250,5,250);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.36

glPushMatrix();

glTranslatef(200,5,250);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.41

glPushMatrix();

glTranslatef(450,10,300);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.42

glPushMatrix();

glTranslatef(400,10,300);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.43

glPushMatrix();

glTranslatef(350,6,300);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.44

glPushMatrix();

glTranslatef(300,5,300);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.45

glPushMatrix();

glTranslatef(250,5,300);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.51

glPushMatrix();

glTranslatef(450,4,350);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.52

glPushMatrix();

glTranslatef(400,3,350);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon2.53

glPushMatrix();

glTranslatef(350,9,350);

glScalef(7,7,7);

pohon2();

glPopMatrix();

//pohon

for (int i = -9; i < -4; i++)

for (int j = -7; j < -2; j++)

{

glPushMatrix();

glTranslatef(i\*50,14,j\*50);

glScalef(7,7,7);

pohon2();

glPopMatrix();

}

for (int i = -6; i < -2; i++)

for (int j = 3; j < 8; j++)

{

glPushMatrix();

glTranslatef(i\*50,5,j\*50);

glScalef(7,7,7);

pohon2();

glPopMatrix();

}

for (int i = 7; i < 10; i++)

for (int j = -7; j < -2; j++)

{

glPushMatrix();

glTranslatef(i\*50,2.5,j\*50);

glScalef(7,7,7);

pohon2();

glPopMatrix();

}

for (int i = 1; i < 10; i++)

for (int j = -2; j < 0; j++)

{

glPushMatrix();

glTranslatef(i\*50,-2.5,j\*20);

glScalef(2,2,2);

pohon3();

glPopMatrix();

}

//pohon4

glPushMatrix();

glTranslatef(120,-2.5,0);

glScalef(5,5,5);

pohon4();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(50,-2.5,0);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(100,-2.5,0);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

for (int i = 1; i < 3; i++)

for (int j = -2; j < 6; j++)

{

glPushMatrix();

glTranslatef(i\*50,-2.5,j\*20);

glScalef(2,2,2);

pohon3();

glPopMatrix();

}

//pohon3

glPushMatrix();

glTranslatef(90,-2.5,120);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(40,-2.5,120);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(80,-2.5,140);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(30,-2.5,140);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(70,-2.5,160);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(20,-2.5,160);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(60,-2.5,180);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon31

glPushMatrix();

glTranslatef(10,-2.5,180);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(50,-2.5,200);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(0,-2.5,200);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(40,-2.5,220);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(-10,-2.5,220);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(30,-2.5,240);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(-20,-2.5,240);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(20,-2.5,260);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(-30,-2.5,260);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(10,-2.5,280);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(-40,-2.5,280);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(0,-2.5,300);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(-50,-2.5,300);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(-10,-2.5,320);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon31

glPushMatrix();

glTranslatef(-60,-2.5,320);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(-20,-2.5,340);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(-70,-2.5,340);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(-30,-2.5,360);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//pohon3

glPushMatrix();

glTranslatef(-80,-2.5,360);

glScalef(2,2,2);

pohon3();

glPopMatrix();

//Sleep(5);

glutSwapBuffers();

}

void reshape(int w, int h) {

float ratio;

ratio = 1.0f \* w / h;

// Reset the coordinate system before modifying

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Set the viewport to be the entire window

glViewport(0, 0, w, h);

// Set the clipping volume

gluPerspective(45,ratio,0.1,1000);

glMatrixMode(GL\_MODELVIEW);

}

// -----------------------------------

// KEYBOARD

// -----------------------------------

void processNormalKeys(unsigned char key, int xx, int yy) {

switch (key) {

case 27:

if (glutGameModeGet(GLUT\_GAME\_MODE\_ACTIVE) != 0)

glutLeaveGameMode();

exit(0);

break;

}

if (key == 'w')

{deltaMove = 10.0f;}

else

if (key == 's')

{deltaMove = -10.0f;}

else

{deltaMove = 0.0f;}

}

void pressKey(int key, int xx, int yy) {

switch (key) {

case GLUT\_KEY\_UP : deltaMove = 10.0f; break;

case GLUT\_KEY\_DOWN : deltaMove = -10.0f; break;

case GLUT\_KEY\_HOME : home += 10.0f; break;

case GLUT\_KEY\_END : home -= 10.0f; break;

case GLUT\_KEY\_PAGE\_UP : home += 0.5f; break;

case GLUT\_KEY\_PAGE\_DOWN : home -= 0.5f; break;

}

if (glutGameModeGet(GLUT\_GAME\_MODE\_ACTIVE) == 0)

sprintf(currentMode,"Current Mode: Window");

else

sprintf(currentMode,

"Current Mode: Game Mode %dx%d at %d hertz, %d bpp",

glutGameModeGet(GLUT\_GAME\_MODE\_WIDTH),

glutGameModeGet(GLUT\_GAME\_MODE\_HEIGHT),

glutGameModeGet(GLUT\_GAME\_MODE\_REFRESH\_RATE),

glutGameModeGet(GLUT\_GAME\_MODE\_PIXEL\_DEPTH));

}

void releaseKey(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_UP :

case GLUT\_KEY\_DOWN : deltaMove = 0;break;

}

}

// -----------------------------------

// MOUSE

// -----------------------------------

void mouseMove(int x, int y) {

// this will only be true when the left button is down

if (xOrigin >= 0) {

// update deltaAngle

deltaAngle = (x - xOrigin) \* 0.01f;

// update camera's direction

lx = sin(angle + deltaAngle);

lz = -cos(angle + deltaAngle);

}

}

void mouseButton(int button, int state, int x, int y) {

// only start motion if the left button is pressed

if (button == GLUT\_LEFT\_BUTTON) {

// when the button is released

if (state == GLUT\_UP) {

angle += deltaAngle;

xOrigin = -1;

}

else {// state = GLUT\_DOWN

xOrigin = x;

}

}

}

void animate()

{

//==============================

if(mv\_c==1)

{

mv+=1;

if(mv==160)

{

mv\_c=0;

}

}

//berotasi\_1 180 derajat

if(mv\_c==0 and mv\_r<=180)

mv\_r+=1;

if(mv\_c==0 and mv\_r==181 and mv\_c1==1)

{

mv1+=1;

if(mv1==160)

{

mv\_c1=0;

}

}

//berotasi\_2 180 derajat

if(mv\_c1==0 and mv\_r1<=180)

mv\_r1+=1;

if(mv\_c1==0 and mv\_r1==181)

{

mv=1;

mv1=1;

mv\_c=1;

mv\_c1=1;

mv\_r=0;

mv\_r1=0;

}

//moving bebek 3

//==============================

if(mva\_c==1 and mva\_c1==1)

{

mva+=1;

if(mva==160)

mva\_c=0;

}

if(mva\_c==0 and mva\_c1==1)

{

mva1+=1;

if(mva1==100)

mva\_c1=0;

}

if(mva\_c==0 and mva\_c1==0)

{

mva2+=1;

if(mva2==45)

{

mva\_c=1;

mva\_c1=1;

mva=1;

mva1=1;

mva2=1;

}

}

// Normaly openGL doesn't continuosly draw frames. It puts one in place and waits for you to tell him what to do next.

// Calling glutPostRedisplay() forces a redraw with the new angle

glutPostRedisplay();

}

void init() {

// register callbacks

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutIdleFunc(animate);

glutIgnoreKeyRepeat(1);

glutKeyboardFunc(processNormalKeys);

glutSpecialFunc(pressKey);

glutSpecialUpFunc(releaseKey);

glutMouseFunc(mouseButton);

glutMotionFunc(mouseMove);

// OpenGL init

/\*glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_CULL\_FACE);\*/

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glDepthFunc(GL\_LESS);

glEnable(GL\_NORMALIZE);

glEnable(GL\_COLOR\_MATERIAL);

glDepthFunc(GL\_LEQUAL);

glShadeModel(GL\_SMOOTH);

glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST);

glEnable(GL\_CULL\_FACE);

\_terrain = loadTerrain("heightmap.bmp", 20);

\_terrainTanah = loadTerrain("heightmapTanah.bmp", 20);

\_terrainAir = loadTerrain("heightmapAir.bmp", 20);

}

int main(int argc, char \*\*argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_STENCIL | GLUT\_DEPTH); //add a stencil buffer to the window

glutInitWindowSize(800, 600);

glutInitWindowPosition(100, 100);

glutCreateWindow("TuBes - Grafkomp");

init();

initRendering();

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

//glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

//glMaterialfv(GL\_FRONT, GL\_SHININESS, high\_shininess);

glColorMaterial(GL\_FRONT, GL\_DIFFUSE);

glutMainLoop();

return 1;

}

**imageloader.cpp**

#include <assert.h>

#include <fstream>

#include "imageloader.h"

using namespace std;

Image::Image(char\* ps, int w, int h) : pixels(ps), width(w), height(h) {

}

Image::~Image() {

delete[] pixels;

}

namespace {

//Converts a four-character array to an integer, using little-endian form

int toInt(const char\* bytes) {

return (int)(((unsigned char)bytes[3] << 24) |

((unsigned char)bytes[2] << 16) |

((unsigned char)bytes[1] << 8) |

(unsigned char)bytes[0]);

}

//Converts a two-character array to a short, using little-endian form

short toShort(const char\* bytes) {

return (short)(((unsigned char)bytes[1] << 8) |

(unsigned char)bytes[0]);

}

//Reads the next four bytes as an integer, using little-endian form

int readInt(ifstream &input) {

char buffer[4];

input.read(buffer, 4);

return toInt(buffer);

}

//Reads the next two bytes as a short, using little-endian form

short readShort(ifstream &input) {

char buffer[2];

input.read(buffer, 2);

return toShort(buffer);

}

//Just like auto\_ptr, but for arrays

template<class T>

class auto\_array {

private:

T\* array;

mutable bool isReleased;

public:

explicit auto\_array(T\* array\_ = NULL) :

array(array\_), isReleased(false) {

}

auto\_array(const auto\_array<T> &aarray) {

array = aarray.array;

isReleased = aarray.isReleased;

aarray.isReleased = true;

}

~auto\_array() {

if (!isReleased && array != NULL) {

delete[] array;

}

}

T\* get() const {

return array;

}

T &operator\*() const {

return \*array;

}

void operator=(const auto\_array<T> &aarray) {

if (!isReleased && array != NULL) {

delete[] array;

}

array = aarray.array;

isReleased = aarray.isReleased;

aarray.isReleased = true;

}

T\* operator->() const {

return array;

}

T\* release() {

isReleased = true;

return array;

}

void reset(T\* array\_ = NULL) {

if (!isReleased && array != NULL) {

delete[] array;

}

array = array\_;

}

T\* operator+(int i) {

return array + i;

}

T &operator[](int i) {

return array[i];

}

};

}

Image\* loadBMP(const char\* filename) {

ifstream input;

input.open(filename, ifstream::binary);

assert(!input.fail() || !"Could not find file");

char buffer[2];

input.read(buffer, 2);

assert(buffer[0] == 'B' && buffer[1] == 'M' || !"Not a bitmap file");

input.ignore(8);

int dataOffset = readInt(input);

//Read the header

int headerSize = readInt(input);

int width;

int height;

switch(headerSize) {

case 40:

//V3

width = readInt(input);

height = readInt(input);

input.ignore(2);

assert(readShort(input) == 24 || !"Image is not 24 bits per pixel");

assert(readShort(input) == 0 || !"Image is compressed");

break;

case 12:

//OS/2 V1

width = readShort(input);

height = readShort(input);

input.ignore(2);

assert(readShort(input) == 24 || !"Image is not 24 bits per pixel");

break;

case 64:

//OS/2 V2

assert(!"Can't load OS/2 V2 bitmaps");

break;

case 108:

//Windows V4

assert(!"Can't load Windows V4 bitmaps");

break;

case 124:

//Windows V5

assert(!"Can't load Windows V5 bitmaps");

break;

default:

assert(!"Unknown bitmap format");

}

//Read the data

int bytesPerRow = ((width \* 3 + 3) / 4) \* 4 - (width \* 3 % 4);

int size = bytesPerRow \* height;

auto\_array<char> pixels(new char[size]);

input.seekg(dataOffset, ios\_base::beg);

input.read(pixels.get(), size);

//Get the data into the right format

auto\_array<char> pixels2(new char[width \* height \* 3]);

for(int y = 0; y < height; y++) {

for(int x = 0; x < width; x++) {

for(int c = 0; c < 3; c++) {

pixels2[3 \* (width \* y + x) + c] =

pixels[bytesPerRow \* y + 3 \* x + (2 - c)];

}

}

}

input.close();

return new Image(pixels2.release(), width, height);

}

**vec3f.cpp**

#include <math.h>

#include "vec3f.h"

using namespace std;

Vec3f::Vec3f() {

}

Vec3f::Vec3f(float x, float y, float z) {

v[0] = x;

v[1] = y;

v[2] = z;

}

float &Vec3f::operator[](int index) {

return v[index];

}

float Vec3f::operator[](int index) const {

return v[index];

}

Vec3f Vec3f::operator\*(float scale) const {

return Vec3f(v[0] \* scale, v[1] \* scale, v[2] \* scale);

}

Vec3f Vec3f::operator/(float scale) const {

return Vec3f(v[0] / scale, v[1] / scale, v[2] / scale);

}

Vec3f Vec3f::operator+(const Vec3f &other) const {

return Vec3f(v[0] + other.v[0], v[1] + other.v[1], v[2] + other.v[2]);

}

Vec3f Vec3f::operator-(const Vec3f &other) const {

return Vec3f(v[0] - other.v[0], v[1] - other.v[1], v[2] - other.v[2]);

}

Vec3f Vec3f::operator-() const {

return Vec3f(-v[0], -v[1], -v[2]);

}

const Vec3f &Vec3f::operator\*=(float scale) {

v[0] \*= scale;

v[1] \*= scale;

v[2] \*= scale;

return \*this;

}

const Vec3f &Vec3f::operator/=(float scale) {

v[0] /= scale;

v[1] /= scale;

v[2] /= scale;

return \*this;

}

const Vec3f &Vec3f::operator+=(const Vec3f &other) {

v[0] += other.v[0];

v[1] += other.v[1];

v[2] += other.v[2];

return \*this;

}

const Vec3f &Vec3f::operator-=(const Vec3f &other) {

v[0] -= other.v[0];

v[1] -= other.v[1];

v[2] -= other.v[2];

return \*this;

}

float Vec3f::magnitude() const {

return sqrt(v[0] \* v[0] + v[1] \* v[1] + v[2] \* v[2]);

}

float Vec3f::magnitudeSquared() const {

return v[0] \* v[0] + v[1] \* v[1] + v[2] \* v[2];

}

Vec3f Vec3f::normalize() const {

float m = sqrt(v[0] \* v[0] + v[1] \* v[1] + v[2] \* v[2]);

return Vec3f(v[0] / m, v[1] / m, v[2] / m);

}

float Vec3f::dot(const Vec3f &other) const {

return v[0] \* other.v[0] + v[1] \* other.v[1] + v[2] \* other.v[2];

}

Vec3f Vec3f::cross(const Vec3f &other) const {

return Vec3f(v[1] \* other.v[2] - v[2] \* other.v[1],

v[2] \* other.v[0] - v[0] \* other.v[2],

v[0] \* other.v[1] - v[1] \* other.v[0]);

}

Vec3f operator\*(float scale, const Vec3f &v) {

return v \* scale;

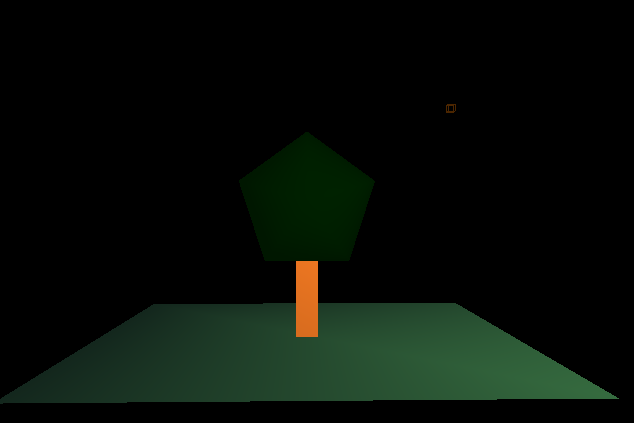
}

ostream &operator<<(ostream &output, const Vec3f &v) {

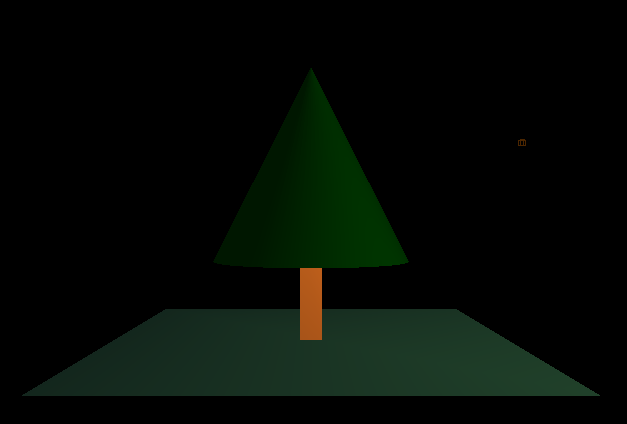
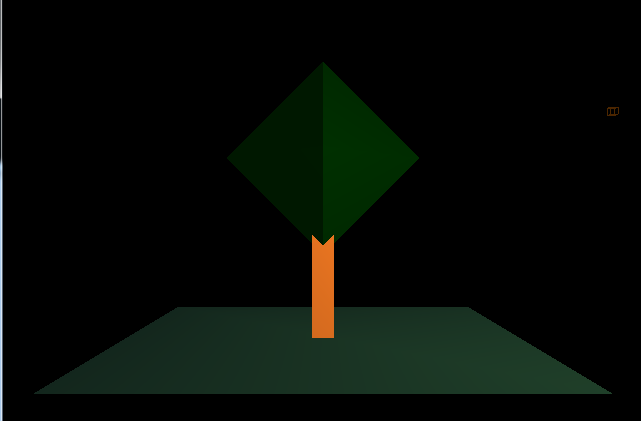
cout << '(' << v[0] << ", " << v[1] << ", " << v[2] << ')';

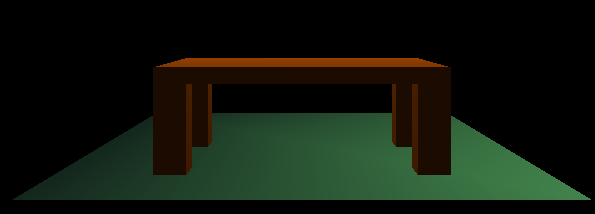
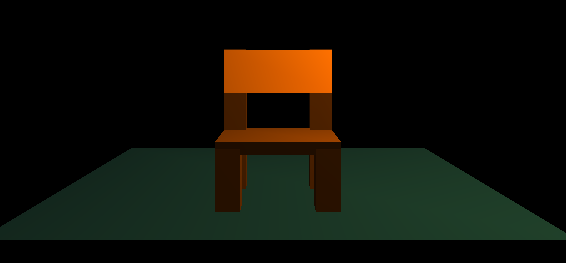
return output;

}

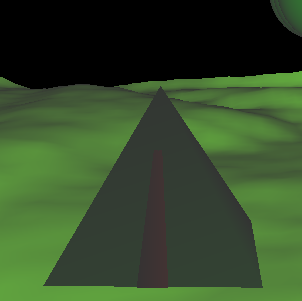
Berikut adalah bentuk-bentuk yang digunakan:

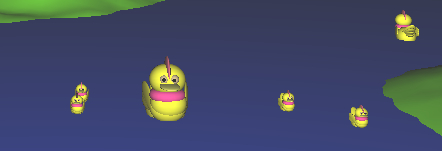
Gambar 3.0 Poho bulat segilima Gambar 3.1 pohon bulat-bulat

Gambar 3.2 Pohon kerucut Gambar 3.4 Pohon octahedron



Gambar 3.5 Meja Gambar 3.6 Kursi

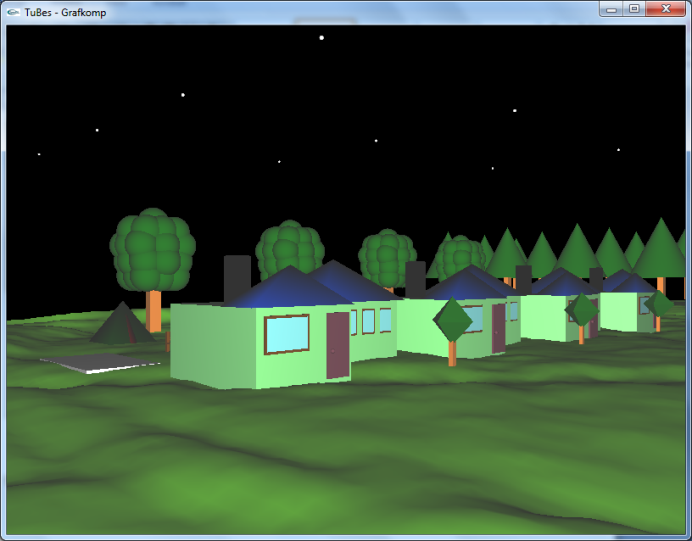
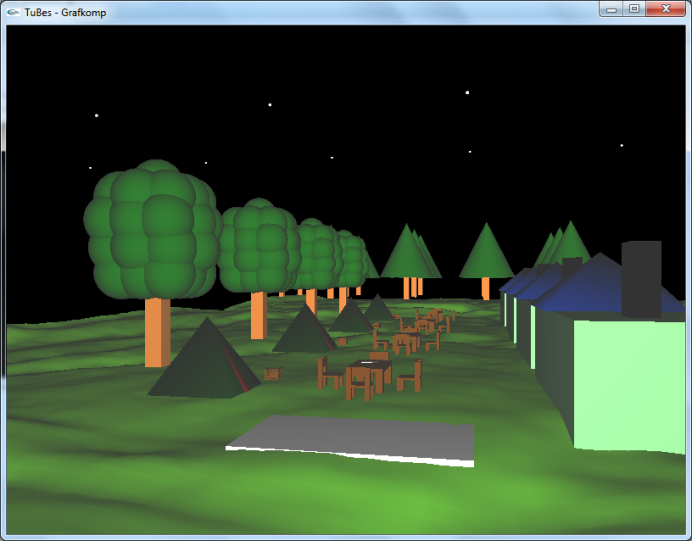


Gambar 3.7 Rumah

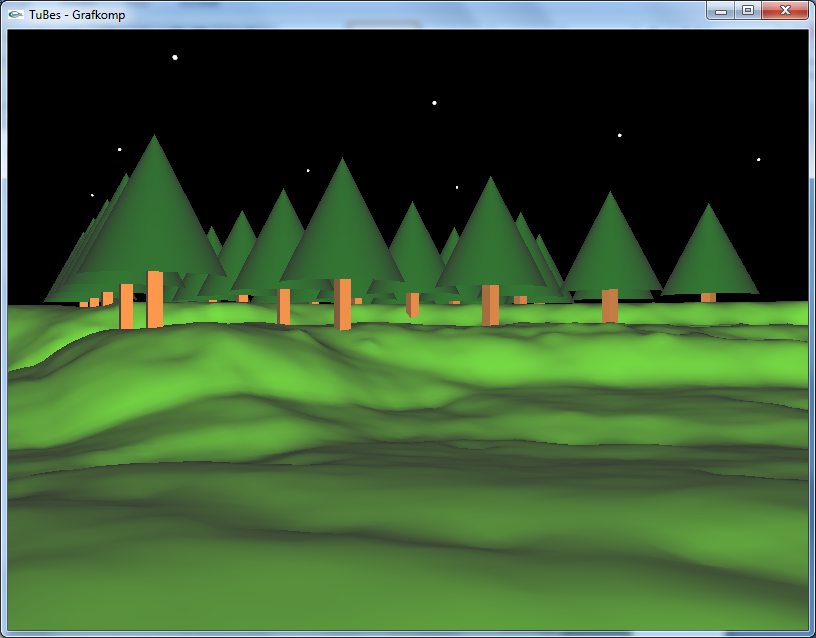
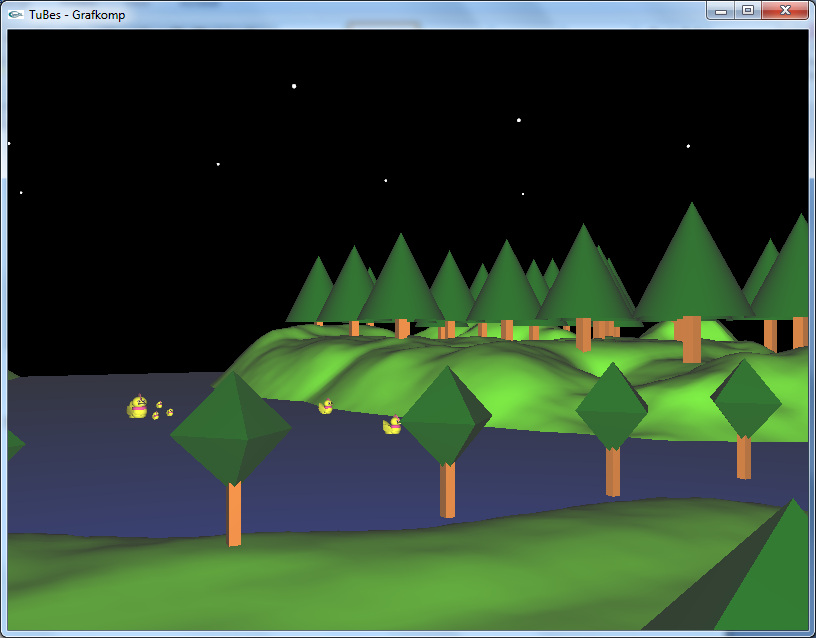
Gambar 3.8 Tenda

Gambar 3.9 Bebek

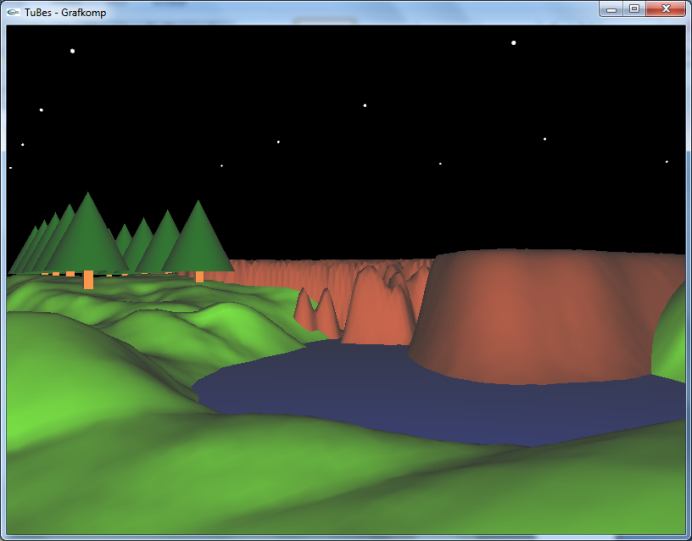
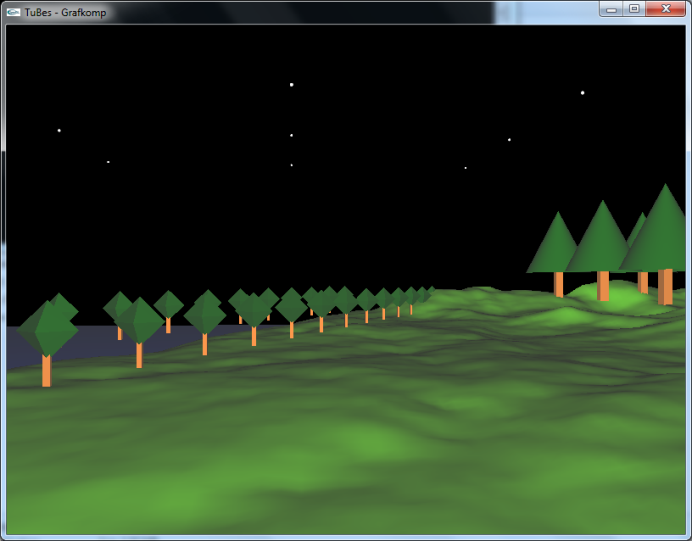
Setelah dicompile dan run, maka hasilnya seperti screenshoot dibawah ini :



Gambar 3.10 Tampilan Belakang Rumah Gambar 3.11 Tampilan Depan Rumah



Gambar 3.12 Tampilan Danau Gambar 3.13 Tampilan Hutan

Gambar 3.14 Tampilan Gunung Gambar 3.15 Tampilan Lapangan

# Bab IV

# Kesimpulan dan saran

Grafika komputer akhir-akhir ini mulai dirasa sangat penting dan mencakup hampir semua bidang kehidupan seiring dengan semakin pentingnya sistem komputer dalam berbagai kegiatan. Grafika komputer merupakan gambar atau grafik yang dihasilkan oleh komputer. Teknik-teknik yang dipelajari dalam grafika komputer adalah teknik-teknik bagaimana membuat atau manciptakan gambar dengan menggunakan komputer, salah satunya adalah Teknik Transformasi tiga dimensi yaitu suatu model atau bentuk atau teknik memindahkan atau mengubah nilai posisi objek dalam sistem koordinat tiga dimensi. Pemindahan objek ini dapat diartikan sebagai pemindahan titik. Transformasi tiga dimensi ini mencakup proses translasi, penskalaan, rotasi, dll.

Dari laporan tugas ini diketahui cara-cara memindahkan polygon dengan GL Translatef, memutar objek dengan GL Rotatef, membuat polygon tersebut lebih besar atau lebih kecil dengan menggunakan GL Scalef. Di samping perintah-perintah yang kami sebutkan di atas, masih banyak lagi objek-objek yang dapat dihasilkan dengan menggunkan transformasi tiga dimensi.

Atas terselesainya laporan tugas ini semoga dapat memberi manfaat bagi yang membaca. Namun, kami juga menyadari bahwa isi laporan tugas ini masih jauh dari sempurna.Untuk waktu dan kesempatan yang diberikan oleh dosen pengajar tidak lupa kami ucapkan banyak terima kasih.