**TUGAS BESAR**

**GRAFIKA KOMPUTER**

**“MEMBANGUN OBJEK 3D GEDUNG SATE BANDUNG”**

Diajukan untuk memenuhi

Tugas Mata Kuliah Grafika Komputer

Program Strata Satu Jurusan Teknik Informatika

Fakultas Teknik dan Ilmu Komputer

Universitas Komputer Indonesia

**OLEH :**

**ANDA ALIMUDIN 10108381**

**ARIEP DWI NUGROHO 10108390**

**ASWIN NURMAN PRADANA 10108412**

**EKO GUNAWAN 10108386**

**WITA MUTIARA 10106451**



**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK DAN ILMU KOMPUTER**

**UNIVERSITAS KOMPUTER INDONESIA**

**BANDUNG**

**2012**

**DAFTAR ISI**

DAFTAR ISI............................................................................................................ i

**BAB I PENDAHULUAN.......................................................................................1**

[1.1 Latar Belakang Masalah 1](#_Toc329072963)

[1.2 Identifikasi Masalah 2](#_Toc329072964)

[1.3 Maksud dan Tujuan 2](#_Toc329072965)

[1.4 Batasan Masalah 3](#_Toc329072966)

**BAB II LANDASAN TEORI................................................................................4**

[2.1 Pengertian Grafika Komputer 4](#_Toc329072967)

[2.2 Elemen – Elemen Dasar Grafika Komputer 5](#_Toc329072968)

[2.2.1 Point 5](#_Toc329072969)

[2.2.2 Polyline 5](#_Toc329072970)

[2.2.3 Polygon 5](#_Toc329072971)

[2.2.4 Filled Polygon (Face) 5](#_Toc329072972)

[2.2.5 Gradate Polygon 6](#_Toc329072973)

[2.3 Grafik Komputer 2D 6](#_Toc329072974)

[2.4 Grafik Komputer 3D 6](#_Toc329072975)

[2.5 Perbedaan Grafik 2D dan Grafik 3D 7](#_Toc329072976)

[2.6 Transformasi 8](#_Toc329072977)

[2.7 *Color* (Warna) 12](#_Toc329072978)

[2.8 Lighting (pencahayaan) 13](#_Toc329072979)

[2.9 Blending (Pencampuran) 14](#_Toc329072980)

[2.10 Texture Mapping 16](#_Toc329072981)

[2.10.1 Menentukan Tekstur Image 16](#_Toc329072982)

[2.10.2 Mendefinisikan gambar sebagai sebuah tekstur 17](#_Toc329072983)

[2.10.3 Mengubah gambar tekstur 17](#_Toc329072984)

[2.10.4 Mapping Tekstur : 18](#_Toc329072989)

[2.10.5 Fog 18](#_Toc329072990)

[2.11 Pengertian OpenGl 18](#_Toc329072991)

[2.12 Visual C++ 19](#_Toc329072992)

**BAB III IMPLEMENTASI.................................................................................21**

[3.1 Source Code Program OpenGL 21](#_Toc329072993)

3.2 Tampilan Program OpenGL………………………………………………65

**BAB IV KESIMPULAN DAN SARAN..............................................................66**

[4.1 Kesimpulan 66](#_Toc329072994)

[4.2 Saran 66](#_Toc329072995)

**BAB I**

**PENDAHULUAN**

* 1. Latar Belakang Masalah

Gedung sate adalah salah satu bangunan tertua di Bandung. Gedung sate dibangun saat masa penjajahan Belanda, sekitar tahun 1920. Awalnya gedung sate diperuntukan bagi Departemen Lalulintas dan Pekerjaan Umum, bahkan menjadi pusat pemerintahan Hindia Belanda setelah Batavia dianggap sudah tidak memenuhi syarat sebagai pusat pemerintahan karena perkembangannya, sehingga digunakan oleh Jawatan Pekerjaan Umum. Gedung Sate merupakan gedung yang memiliki sejarah tinggi bagi kota Bandung, untuk itu gedung sate menjadi bahan pembelajaran bagi masyarakat.

Pemodelan bangunan tidak terlepas dari sebuah grafis 3D, agar *user* tidak merasa bosan untuk melihat model tersebut, serta *user* akan tertarik mempelajari dari struktur bangunan tersebut. Banyak cara untuk memodelkan bangunan, salah satunya menggunakan OpenGL. OpenGL adalah spesifikasi standar yang mendefinisikan sebuah lintas-bahasa, lintas platform API untuk mengembangkan aplikasi yang menghasilkan grafis komputer 2D maupun3D.

Untuk itu memerlukan pemodelan dari Gedung Sate sebagai bahan pembelajaran serta sebagai model untuk merepresentasikan struktur bangunan untuk bahan pengembangannya menggunakan OpenGL. Dalam pembuatannya, objek gedung sate dibuat dengan fungsi tiga dimensi yang terdapat pada OpenGL. Menerapkan penempatan-penempatan vertex untuk membangun gedung dan akan diberikan nilai rotasi agar terlihat dari sudut pandang manapun. Pada program yang dibuat pun akan diberikan fungsi keyboard untuk mempermudah *user* dalam penggunaan program tersebut.

* 1. Identifikasi Masalah

Berdasarkan uraian latar belakang diatas dapat diidentifikasikan masalah yaitu :

1. Bagaimana cara memodelkan bangunan-bangunan tua yang memiliki nilai sejarah tinggi agar masyarakat dapat mengetahuinnya?
2. Keingintahuan masyarakat terhadap model gedung sate menjadikan perlunya pemodelan grafis untuk mengilustrasikan terhadap model yang sebenarnya.
   1. Maksud dan Tujuan

Berdasarkan permasalahan yang diteliti, maka maksud dari penulisan tugas besar ini adalah untuk membangun aplikasi pemodelan gedung sate dengan berbasis desktop. Sedangkan tujuan yang akan dicapai adalah :

1. Membangun aplikasi untuk memodelkan suatu tempat yang memiliki nilai sejarah tinggi.
2. Membangun suatu pemodelan untuk memodelkan bangunan Gedung Sate.
3. Mengaplikasikan materi openGL dari mata kuliah Grafika Komputer.
   1. Batasan Masalah

Karena masalah yang harus dibahas luas, maka batasan-batasan adalah sebagai berikut:

1. Pembangunan pemodelan tersebut menerapkan bahasa OpenGL.
2. Pemodelan akan menerapkan konsep 3 Dimensi.
3. Pemodelan hanya akan memodelkan bentuk dari bangunan objek saja (Gedung Sate).

**BAB II**

**LANDASAN TEORI**

* 1. 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.

* 1. Elemen – Elemen Dasar Grafika Komputer

Ada beberapa elemen dasar dari grafika komputer antara lain:

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

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

* + 1. Polyline

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

* + 1. Polygon

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

* + 1. 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.

* + 1. Gradate Polygon

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

* 1. 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.

* 1. 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.

* 1. Perbedaan Grafik 2D dan Grafik 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.

* 1. 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**

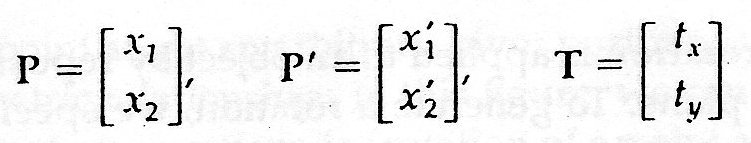
Translasi dilakukan dengan penambahan translasi pada suatu titik koordinat dengan translasi vektor atau shift vektor, yaitu (tx,ty), dimana tx adalah translation vektor menurut sumbu x, sedangkan ty adalah translation vektor menurut sumbu y. koordinat baru titik yang ditranslasi dapat diperoleh dengan

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.

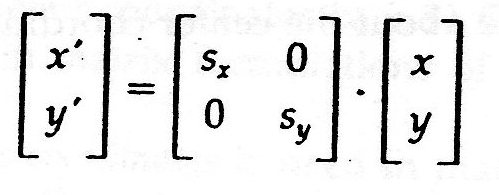
1. **Skala**

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:

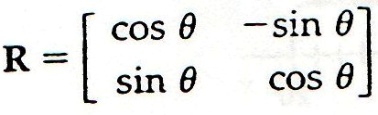
x’ = x cos θ – y sin θ

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

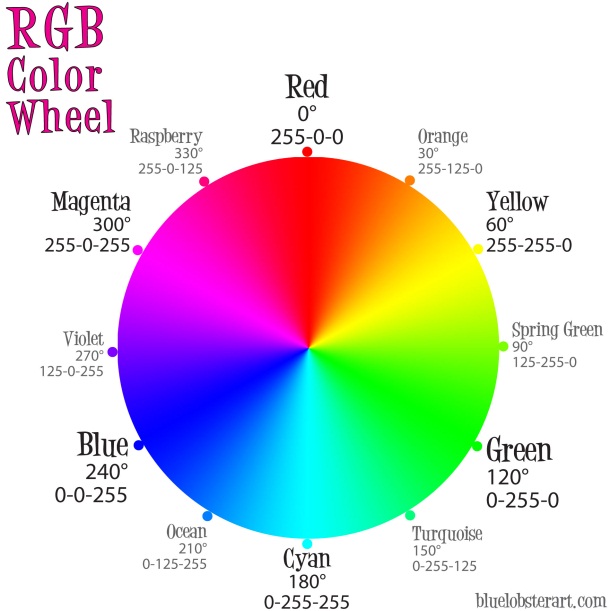
* 1. *Color* (Warna)

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

**Pembagian Warna**

1. RGB (Red-Green-Blue) : warna dasar yang dijadikan patokan warna secara universal (primary colors)
2. 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 2.1 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.
   1. 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.

* 1. 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].

* 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
   1. Membaca atau membangkitkan tekstur
   2. Menandai tekstur
   3. Mengenablekan tekstur
2. Menandai koordinat tekstur pada vertek
3. Menentukan parameter tekstur
   1. Wrapping , filtering, dsb.

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

* + 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
   1. Gambar yang didefinisikan (baik secara manual maupun dengn suatu fungsi matematik tertentu).
   2. Membangkitkan dengan kode aplikasi
3. Mengenablekan tekstur mapping
   1. glEnable(GL\_TEXTURE\_2D)
   2. OpenGL mendukung 1 sampai 4 dimensional tekstur mapping
      1. Mendefinisikan gambar sebagai sebuah tekstur

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);

* + 1. 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
   * 1. Mapping Tekstur :
5. Didasarkan pada koordinat tekstur parametric
6. glTexCoord\*() ditetapkan pada masing – masing vertex
   * 1. 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

* 1. Pengertian OpenGl

OpenGL adalah API (Application Programing Interface) yang dikenalkan oleh SGI (Silicon Graphics Inc) kali pertama pada tahun 1980-an. Awalnya OpenGL diciptakan untuk mendukung proses rendering, pada saat itu hanya digunakan untuk mendukung proses redering yang dilakukan oleh komputer graphic produksi SGI. Namun akhirnya, OpenGL dijadikan standar oleh berbagai perusahaan software dan hardware.

open GL (program java for openGL). adalah bahasa yang dapat dijalankan dimanapun dan di sembarang platform apapun, diberagam lingkungan : internet, intranets, consumer electronic products, dan computer applications. Bahasa pemrograman berorientasi objek telah menjadi aliran utama (mainstream), java benar-benar berorientasi objek sejati, melebihi C++.Segala sesuatu dijava kecuali sedikit type dasar (int, float, double, char) adalah objek.

* 1. Visual C++

Visual C++ adalah sebuah produk [Integrated Development Environment](http://id.wikipedia.org/wiki/Integrated_Development_Environment) (IDE) untuk [bahasa pemrograman](http://id.wikipedia.org/wiki/Bahasa_pemrograman)[C](http://id.wikipedia.org/wiki/Bahasa_pemrograman_C) dan [C++](http://id.wikipedia.org/wiki/C%2B%2B) yang dikembangkan [Microsoft](http://id.wikipedia.org/wiki/Microsoft). Visual C++ merupakan salah satu bagian dari paket [Microsoft](http://id.wikipedia.org/wiki/Microsoft)[Visual Studio](http://id.wikipedia.org/wiki/Visual_Studio).Bahasa C atau C++ adalah suatu bahasa pemrograman.Bahasa C termasuk sebagai bahasa pemrograman tingkat menengah, maksudnya bahasa C bisa dipelajari dengan lebih mudah karena mudah dimengerti tetapi mempunyai kemampuan yang tinggi.

Bahasa C bisa digunakan untuk merekayasa program untuk segala kebutuhan, baik untuk aplikasi bisnis, matematis atau bahkan game.Semua bahasa mempunyai kelemahan atau kelebihan sendiri-sendiri. Begitu juga dengan bahasa C. Adapun sebagian kelebihan dari bahasa C adalah sebagai berikut :

1. Banyak memiliki operator untuk mengolah / memanipulasi data.
2. Bahasa C termasuk sebagai bahasa yang terstruktur sehingga program dapat
3. lebih mudah dipahami atau dikembangkan.
4. Bahasa C lebih mudah dimengerti karena lebih mirip kepada bahasa manusia.
5. Kecepatan eksekusi tinggi.
6. Mengenal data pointer.
7. Sedangkan kelemahan dari bahasa C adalah :
8. Banyaknya operator atau cara penulisan program kadang menimbulkan
9. kebingungan para pemakainya.
10. Perlunya ketelitian dalam penulisan program karena perintah (statement)
11. dalam bahasa C bersifat case sensitiv (huruf kapital dan huruf kecil dibedakan

**BAB III**

**IMPLEMENTASI**

* 1. Source Code Program OpenGL

|  |
| --- |
| //GedungSate.cpp  #include <iostream>  #include "stdlib.h"  #include "gl/glut.h"  #include "math.h"  #include "imageloader.h"  #include "vec3f.h"  using namespace std;  int w=600, h=600, z=10;  int a=0 , b=0, c=0, d=0;  int x1=0, y2=0, sudut=0, z1=0;  float skalaX=1, skalaY=1, skalaZ=1;  int cx, cy;  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[] = { 70.0f };  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];  }  };  GLuint loadTexture(Image\* image) {  GLuint textureId;  glGenTextures(1, &textureId); //Make room for our texture  glBindTexture(GL\_TEXTURE\_2D, textureId); //Tell OpenGL which texture to edit  //Map the image to the texture  glTexImage2D(GL\_TEXTURE\_2D, //Always GL\_TEXTURE\_2D  0, //0 for now  GL\_RGB, //Format OpenGL uses for image  image->width, image->height, //Width and height  0, //The border of the image  GL\_RGB, //GL\_RGB, because pixels are stored in RGB format  GL\_UNSIGNED\_BYTE, //GL\_UNSIGNED\_BYTE, because pixels are stored  //as unsigned numbers  image->pixels); //The actual pixel data  return textureId; //Returns the id of the texture  }  GLuint \_textureId;  //Loads a terrain from a heightmap. The heights of the terrain range from  //-height / 2 to height / 2.  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) - 0.5f);  t->setHeight(x, y, h);  }  }    delete image;  t->computeNormals();  return t;  }  float \_angle = 60.0f;  Terrain\* \_terrain;  void cleanup() {  delete \_terrain;  }  void myKeyboard(unsigned char key, int x, int y){  if (key =='a') z+=5;  else if (key == 'd') z-=5;  else if (key == 'w') {  x1=0;  y2=-1;  z1=0;  sudut+=-10;  }  else if (key == 's') {  y2=1;  x1=0;  z1=0;  sudut+=-10;  }  else if (key == 'q') {  y2=0;  x1=0;  z1=1;  sudut+=-10;  }  else if (key=='z'){  a+=-1;  d+=1;  b=0;  c=0;  ;  }  else if (key=='c'){  d+=-1;  a+=1;  b=0;  c=0;  ;  }  }  void init(){  glShadeModel(GL\_SMOOTH);  glShadeModel(GL\_SMOOTH);  glShadeModel(GL\_SMOOTH);  glEnable(GL\_NORMALIZE);  glEnable(GL\_COLOR\_MATERIAL);  glClearColor(0.0f,0.0f,0.0f,0.0f);  glClearDepth(1.0f);  glEnable(GL\_DEPTH\_TEST);  glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST);  Image\* image = loadBMP("wall.bmp");  \_textureId = loadTexture(image);  delete image;  glEnable(GL\_LIGHTING);  glEnable(GL\_LIGHT0);  return;  }  void renderScene(void){  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glClearColor(0.0, 1.8, 5.0, 0.0);  glLoadIdentity();  glTranslatef(10,0,z-80);  glRotatef(sudut,x1,y2,z1);  glScalef(skalaX, skalaY, skalaZ);    glPushMatrix();  float scale = 80.0f / max(\_terrain->width() - 1, \_terrain->length() - 1);  glScalef(scale, scale, scale);  glTranslatef(-(float)(\_terrain->width() - 1) / 2,  0.0f,  -(float)(\_terrain->length() - 1) / 2);  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.1f, 0.9f, 0.0f);  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();  } glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();      //tembok  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glRotatef(50,0,1,0);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok2  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,-0.9,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok2  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,-0.9,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok2  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,-0.9,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);        //tembok tangga kanan  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(1,0.7,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);        //tembok4 panjang kanan  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(2,-0.9,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok4 panjang kanan  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(3,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);        //tembok tangga kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-7,1,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-2,-0.9,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-3,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);  //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-3,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(20,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,-2,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //tembok panjang kiri  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);      //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(12,5,1.3);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,-2.6);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-4,-3,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(17,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela atas  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,-2,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //Jendela bawah  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,-0.5,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //pintu    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(-4,0.5,0);  glRotatef(360,360,360,360);  glutSolidCube(3);  glDisable(GL\_COLOR\_MATERIAL);    //pintu    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.8,0,0.2);  glTranslatef(0,1,0);  glRotatef(360,360,360,360);  glutSolidCube(2);  glDisable(GL\_COLOR\_MATERIAL);      //pintu    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,-1.5,-1);  glRotatef(360,360,360,360);  glutSolidCube(2);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bawah kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-5,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bawah kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bawah kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bawah kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bawah kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bawah kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0.5,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      //tembok kanan bawah    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(2,0,-6);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //tembok kanan bawah    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //tembok kanan bawah    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //tembok kanan bawah    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);      //tembok kanan bawah    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);      //tembok kanan bawah    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //tembok kanan bawah    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //jendela bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-0.5,0.7,-1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      //jendela bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(1,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);        //jendela bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0.5,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //jendela bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      //jendela bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);        //jendela bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);        //jendela bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      //jendela bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);        //jendela bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //pintu bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,-1,-3);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //pintu bangunan kiri    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,-0.3,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //tembok bangunan kanan      glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(22,0.5,4);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //tembok bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //tembok bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //tembok bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //tembok bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //tembok bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //tembok bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2,2,2);  glTranslatef(0,0,-1);  glRotatef(360,360,360,360);  glutSolidCube(2.5);  glDisable(GL\_COLOR\_MATERIAL);    //jendela bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-0.7,0.7,-1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //jendela bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(1.2,0,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);      //jendela bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(-1.5,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    //jendela bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //jendela bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,0,1);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //pintu bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,-1,-4);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);  //pintu bangunan kanan    glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.3,0.0,0.0);  glTranslatef(0,-0.6,0);  glRotatef(360,360,360,360);  glutSolidCube(0.6);  glDisable(GL\_COLOR\_MATERIAL);    // Draw ground  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.0f, 0.0f, 0.0f);  glTranslatef(-11,0,0);  glBegin(GL\_QUADS);  glVertex3f( -3.0f, -0.5f, -7.0f);  glVertex3f(-3.0f, -0.5f, 12.0f);  glVertex3f( 3.0f, -0.5f, 12.0f);  glVertex3f( 3.0f, -0.5f, -7.0f);  glEnd();  glDisable(GL\_COLOR\_MATERIAL);    //jalan  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.0f, 0.0f, 0.0f);  glTranslatef(0,-1.9,-20);  glBegin(GL\_QUADS);  glVertex3f( -22.0f, -0.5f, -5.0f);  glVertex3f(-22.0f, -0.5f, 5.0f);  glVertex3f( 22.0f, -0.5f, 5.0f);  glVertex3f( 22.0f, -0.5f, -5.0f);  glEnd();  glDisable(GL\_COLOR\_MATERIAL);      glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2.0f, 2.0f, 2.0f);  glTranslatef(-20,0.2,-1);  glBegin(GL\_QUADS);  glVertex3f(0.0f, -0.5f, 0.0f);  glVertex3f(0.0f, -0.5f, 1.0f);  glVertex3f( 5.0f, -0.5f, 1.0f);  glVertex3f(5.0f, -0.5f, 0.0f);  glEnd();  glDisable(GL\_COLOR\_MATERIAL);    glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2.0f, 2.0f, 2.0f);  glTranslatef(9,0,-0);  glBegin(GL\_QUADS);  glVertex3f(0.0f, -0.5f, 0.0f);  glVertex3f(0.0f, -0.5f, 1.0f);  glVertex3f( 5.0f, -0.5f, 1.0f);  glVertex3f(5.0f, -0.5f, 0.0f);  glEnd();  glDisable(GL\_COLOR\_MATERIAL);    glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2.0f, 2.0f, 2.0f);  glTranslatef(9,0,-0);  glBegin(GL\_QUADS);  glVertex3f(0.0f, -0.5f, 0.0f);  glVertex3f(0.0f, -0.5f, 1.0f);  glVertex3f( 5.0f, -0.5f, 1.0f);  glVertex3f(5.0f, -0.5f, 0.0f);  glEnd();  glDisable(GL\_COLOR\_MATERIAL);    glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2.0f, 2.0f, 2.0f);  glTranslatef(9,0,-0);  glBegin(GL\_QUADS);  glVertex3f(0.0f, -0.5f, 0.0f);  glVertex3f(0.0f, -0.5f, 1.0f);  glVertex3f( 5.0f, -0.5f, 1.0f);  glVertex3f(5.0f, -0.5f, 0.0f);  glEnd();  glDisable(GL\_COLOR\_MATERIAL);    glEnable(GL\_COLOR\_MATERIAL);  glColor3f(2.0f, 2.0f, 2.0f);  glTranslatef(9,0,-0);  glBegin(GL\_QUADS);  glVertex3f(0.0f, -0.5f, 0.0f);  glVertex3f(0.0f, -0.5f, 1.0f);  glVertex3f( 5.0f, -0.5f, 1.0f);  glVertex3f(5.0f, -0.5f, 0.0f);  glEnd();  glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();  //mobil merah    glEnable(GL\_COLOR\_MATERIAL);  glColor3f(1.0f, 0.0f, 0.0f);  glTranslatef(a-20,b-6.5,c-28);  glutSolidCube(1.2);  glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();    //mobil biru  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.0f, 0.0f, 1.0f);  glTranslatef(d+20,b-7.5,c-34);  glutSolidCube(1.2);  glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();    //^^^^update bangunan dari sini^^^  //jendela  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(1.5,1.8,1.5);  glTranslatef(0.5,0.1,1.46);  glScalef(3,3,1);  glutSolidCube(0.01);  glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(1.5,1.8,1);  glTranslatef(0.9,0.1,1.46);  glScalef(3,3,1);  glutSolidCube(0.01);  glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(1.5,1.8,1);  glTranslatef(0.9,-0.3,1.46);  glScalef(3,3,1);  glutSolidCube(0.01);  glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(1.5,1.8,1);  glTranslatef(0.5,-0.3,1.46);  glScalef(3,3,1);  glutSolidCube(0.01);  glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();  //atap  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.8,0,0);  glRotatef(5,0,1,0);  glTranslatef(0,1,0);  glScalef(3,1.3,3);  glutSolidOctahedron();  glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();    //atap2  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.8,0,0);  glRotatef(5,0,1,0);  glTranslatef(0,1.5,0);  glScalef(3,1.3,3);  glutSolidOctahedron();  glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();    //atap3  glPushMatrix();  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.8,0,0);  glRotatef(5,0,1,0);  glTranslatef(0,2,0);  glScalef(3,1.3,3);  glutSolidOctahedron();  glDisable(GL\_COLOR\_MATERIAL);  glPopMatrix();    //sate1  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.8,0,0);  glRotatef(5,0,1,0);  glTranslatef(0,3,0);  glScalef(3,5.3,3);  glutSolidSphere(0.1,12,16);  glDisable(GL\_COLOR\_MATERIAL);    //sate2  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.5,0,0);  glRotatef(5,0,1,0);  glTranslatef(0,0.05,0);  glScalef(3,8,3);  glutSolidSphere(0.02,12,12);  glDisable(GL\_COLOR\_MATERIAL);  //sate3  glEnable(GL\_COLOR\_MATERIAL);  glColor3f(0.5,0,0);  glRotatef(5,0,1,0);  glTranslatef(0,0.02,0);  glScalef(3,7,3);  glutSolidSphere(0.003,12,12);  glDisable(GL\_COLOR\_MATERIAL);    glutSwapBuffers();  }  void resize(int w1, int h1){  glViewport(0,0,w1,h1);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0,(float) w1/(float) h1, 1.0,300.0);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  }  void timer(int value){  glutPostRedisplay();  glutTimerFunc(50,timer,0);  }  main (int argc, char \*\*argv){  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_DEPTH | GLUT\_RGBA);  glutInitWindowPosition(140,60);  glutInitWindowSize(1024,700);  glutCreateWindow("Gedung Sate Jaman Doeloe");  glutDisplayFunc(renderScene);    \_terrain = loadTerrain("heightmap.bmp",13);    glutReshapeFunc(resize);  glutKeyboardFunc(myKeyboard);  glutTimerFunc(1,timer,0);  init();    glutMainLoop();  } |

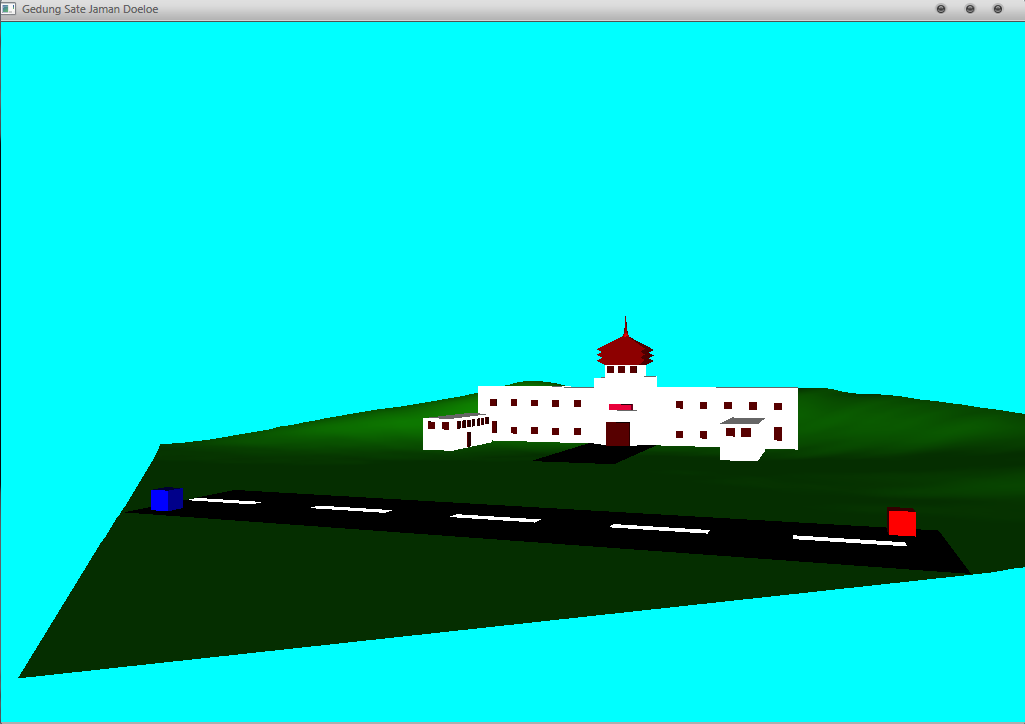
|  |
| --- |
| //imageloader.h  #ifndef IMAGE\_LOADER\_H\_INCLUDED  #define IMAGE\_LOADER\_H\_INCLUDED  //Represents an image  class Image {  public:  Image(char\* ps, int w, int h);  ~Image();    /\* An array of the form (R1, G1, B1, R2, G2, B2, ...) indicating the  \* color of each pixel in image. Color components range from 0 to 255.  \* The array starts the bottom-left pixel, then moves right to the end  \* of the row, then moves up to the next column, and so on. This is the  \* format in which OpenGL likes images.  \*/  char\* pixels;  int width;  int height;  };  //Reads a bitmap image from file.  Image\* loadBMP(const char\* filename);  #endif |

|  |
| --- |
| //vec3f.h  #ifndef VEC3F\_H\_INCLUDED  #define VEC3F\_H\_INCLUDED  #include <iostream>  class Vec3f {  private:  float v[3];  public:  Vec3f();  Vec3f(float x, float y, float z);    float &operator[](int index);  float operator[](int index) const;    Vec3f operator\*(float scale) const;  Vec3f operator/(float scale) const;  Vec3f operator+(const Vec3f &other) const;  Vec3f operator-(const Vec3f &other) const;  Vec3f operator-() const;    const Vec3f &operator\*=(float scale);  const Vec3f &operator/=(float scale);  const Vec3f &operator+=(const Vec3f &other);  const Vec3f &operator-=(const Vec3f &other);    float magnitude() const;  float magnitudeSquared() const;  Vec3f normalize() const;  float dot(const Vec3f &other) const;  Vec3f cross(const Vec3f &other) const;  };  Vec3f operator\*(float scale, const Vec3f &v);  std::ostream &operator<<(std::ostream &output, const Vec3f &v);  #endif |

|  |
| --- |
| //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;  } |

* 1. Tampilan Program OpenGL



Gambar 3.1 Tampilan Program

**BAB IV**

**KESIMPULAN DAN SARAN**

* 1. Kesimpulan

Berdasarkan permasalahan yang telah diteliti, maka kesimpulan dari penulisan tugas besar ini adalah :

1. Dapat membangun aplikasi untuk memodelkan suatu tempat yang memiliki nilai sejarah tinggi.
2. Dapat membangun suatu pemodelan untuk memodelkan bangunan Gedung Sate.
3. Juga dapat memahami dan mengaplikasikan materi openGL dari mata kuliah Grafika Komputer dengan baik.
   1. Saran

Besar harapan kami agar saran dan kritik dari pembaca dapat membangun dan memperbaiki segala kekurangan, agar kami dapat menghasilkan projek yang lebih baik lagi.