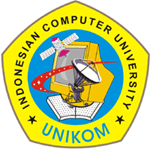
**APLIKASI 3D DIMENSI SATELITE LUAR ANGKASA**

*Makalah Ini Dibuat untuk Memenuhi Tugas Besar Mata Kuliah Komputer Grafika*

**Dosen : Hendri Karisma, S. Kom.**

Disusun Oleh *:*

|  |  |
| --- | --- |
| **A Baso Lolo S** | **10108628** |
| **Muhamad Edwar S** | **10108609** |
| **Raditya Fajar** | **10108611** |
| **Fery Priyandi** | **10108606** |
| **Widi Linggih Jaelani** | **10108641** |
|  |  |



**JURUSAN TEKNIK INFORMATIKA  
FAKULTAS TEKNIK DAN ILMU KOMPUTER  
UNIVERSITAS KOMPUTER INDONESIA  
BANDUNG  
2012**

# KATA PENGANTAR

*Bismillahirohmanirrohim*

Segala puji dan syukur yang tak terhingga penulis panjatkan kehadirat Allah SWT, yang selalu memberikan rahmat dan karunia-Nya kepada penulis sehingga proposal ini dapat selesai.

Laporan yang berjudul **” *APLIKASI 3D DIMENSI SATELITE LUAR ANGKASA”*.** Laporan ini juga dibuat untuk memenuhi tugas besar mata kuliah Komputer Grafika semester 8.

Penulis menyadari sepenuhnya laporan ini masih jauh dari sempurna, oleh karena itu penulis sangat mengharapkan kritik dan saran yang bersifat membangun untuk perbaikan di masa yang akan datang.

*Billahitaufiq Walhidayah, Wassalamu’alaikum Wr. Wb.*

Bandung, Juni 2012

Penulis

# DAFTAR ISI

[KATA PENGANTAR i](#_Toc265697970)

[DAFTAR ISI ii](#_Toc265697971)

[BAB I PENDAHULUAN 1](#_Toc265697972)

[1.1 Latar Belakang Masalah 1](#_Toc265697973)

[1.2 Perumusan masalah 2](#_Toc265697974)

[1.3 Maksud dan Tujuan 2](#_Toc265697975)

[1.4 Batasan Masalah 2](#_Toc265697976)

[BAB II LANDASAN TEORI 3](#_Toc265697977)

[2.1 Visual C++ 3](#_Toc265697978)

[2.2 Sekilas tentang OpenGL 3](#_Toc265697979)

[2.3 Objek 3 Dimensi 6](#_Toc265697980)

[2.4 Bitmap 6](#_Toc265697981)

[2.5 Lighting (Diffuse, Ambient dan Specular) 7](#_Toc265697982)

[2.6 *Mapping* (Pemetaan) 8](#_Toc265697983)

[2.7 *Perspective* 10](#_Toc265697984)

[2.8 *Color* and *Shading* 11](#_Toc265697985)

[2.9 *Blending* (Pencampuran) 12](#_Toc265697986)

[2.10 *Antialiasing* 12](#_Toc265697987)

[2.11 *Translation* 13](#_Toc265697988)

[2.12 *Rotation* 13](#_Toc265697989)

[2.13 *Scaling* 14](#_Toc265697990)

[BAB III ANALISIS DAN PERANCANGAN 15](#_Toc265697991)

[3.1 Analisis Sistem 15](#_Toc265697992)

[3.1.1 Analisis Kebutuhan Non-Fungsional 15](#_Toc265697993)

[3.1.2 Analisis Kebutuhan Fungsional 16](#_Toc265697994)

[3.2 Perancangan 21](#_Toc265697995)

[3.2.1 Perancangan *Interface* (Antarmuka) 22](#_Toc265697996)

[3.2.2 Perancangan Perintah Melalui Keyboard 22](#_Toc265697997)

[BAB IV IMPLEMENTASI 24](#_Toc265697998)

[4.1 Tampilan Awal 24](#_Toc265697999)

[4.2 Tampilan dari beberapa *capture* 25](#_Toc265698000)

[BAB V KESIMPULAN 2](#_Toc265698001)8

[DAFTAR PUSKATA](#_Toc265698002) ........................................................................................................................................... 29

[KONTRIBUSI](#_Toc265698003)

[LAMPIRAN](#_Toc265698004)

# 

# PENDAHULUAN

## Latar Belakang Masalah

Komputer telah berkembang menjadi salah satu media yang dapat diandalkan dalam kegiatan memanipulasi gambar atau grafik. Karena prinsip dari teknologi pengolahan grafis dengan komputer dapat berperan dalam berbagai bidang, maka tidak heran jika pemanfaatan pengolahan grafis dengan komputer ini kian marak dikembangkan. Bahkan pemanfaatan teknologi tersebut telah menghasilkan suatu disiplin ilmu yang tersendiri yaitu grafika komputer.

Definisi dari grafika komputer adalah sekumpulan alat terdiri dari *hardware* dan *software* yang digunakan untuk membuat gambar dan berinteraksi dengan gambar grafik atau citra realistik untuk seni, *game* komputer, foto dan animasi komputer dengan cara-cara seperti yang biasa digunakan. Suatu proses pembuatan, penyimpanan dan manipulasi model, citra atau gambar berdasarkan deskripsi obyek maupun latar belakang yang terkandung pada gambar tersebut. Dimana model berasal dari beberapa bidan seperti fisik, matematik, artistik dan bahkan struktur abstrak

Pemanfaatan grafika komputer tersebut telah menghasilkan beberapa teknologi baru. Salah satunya adalah OpenGL. Sebuah antarmuka pemograman yang digunakan dalam membuat grafik *real-time* 3 dimensi .OpenGLadalah API yang dikenalkan oleh SGI (*Silicon Graphics Inc*) kali pertama pada tahun 1980-an. Awalnya OpenGL diciptakan untuk mendukung proses rendering pada hanya digunakan untuk mendukung proses redering yang dilakukan oleh komputer graphic produksi SGI. Namun akhirnya, OpenGL dijadikan standar oleh berbagai perusahaan *software* dan *hardware*. OpenGL bersifat multi-platform yaitu dapat digunakan hampir pada semua operating system, mulai dari Windows, Linux, sampai Mac OS X. Pemanfaatan OpenGL tersebut telah banyak dimanfaatkan dalam berbagai aplikasi perangkat lunak yang memanfaatkan tampilan atau aimasi 3D. Salah satu contoh *Game* 3D yang menggunakan OpenGL sebagai API adalah Quake.

Oleh karena itu kami bermaksud untuk membuat sebuah aplikasi yang menerapkan konsep-konsep dalam grafika komputer dengan memanfaatkan OpenGL. Pembuatan aplikasi ini diharapkan dapat menambah pemahaman kami tentang grafika komputer dan OpenGL.

## Perumusan masalah

Permasalahan yang akan di bahas adalah :

* Bagaimana mengimplementasikan konsep-konsep grafika komputer dan OpenGL kedalam sebuah aplikasi 3D.
* Bagaimana mengimplementasikan fungsi-fungsi OpenGL yang berhubungan dengan konsep-konsep grafika komputer kedalam sebuah aplikasi 3D.

## Maksud dan Tujuan

Berdasarkan latar belakang masalah yang telah diuraikan diatas, maka kami bermaksud untuk membuat sebuah aplikasi yang menerapkan konsep-konsep dalam grafika komputer dengan memanfaatkan OpenGL.

* Tujuan dari pembuatan aplikasi adalah :

1. Memenuhi tugas pengganti UAS mata kuliah Grafika Komputer.
2. Mengimplementasikan konsep-konsep grafika komputer dan OpenGL dalam sebuah aplikasi animasi 3D,
3. Memahami konsep-konsep grafika komputer dan penerapannya dalam OpenGL secara praktek.

## Batasan Masalah

Teknik yang diimplementasikan hanya meliputi :

* *Perspective*
* *Color & Shading*
* *Lighting (Diffuse, Ambient dan Specular)*
* *Translation, Rotation dan Scaling*
* *Antialiasing*
* *Blending*
* *Mapping Texture*
* Menggunakan OpenGL dan Visual C++ 6.0

# LANDASAN TEORI

## 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 :

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

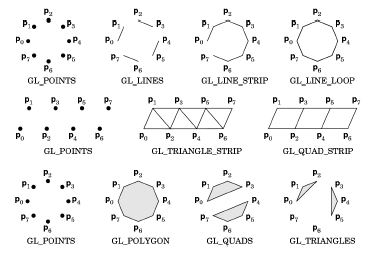
## Sekilas tentang OpenGL

OpenGL adalah suatu spefikasi grafik yang *low-level* yang menyediakan fungsi untuk pembuatan grafik primitif termasuk titik, garis, dan lingkaran. OpenGL digunakan untuk mendefinisikan suatu objek, baik objek 2 dimensi maupun objek 3 dimensi. OpenGL juga merupakan suatu antarmuka pemrograman aplikasi *application programming interface* (API) yang tidak tergantung pada piranti dan platform yang digunakan, sehingga OpenGL dapat berjalan pada sistem operasi Windows, UNIX dan sistem operasi lainnya. OpenGL pada awalnya didesain untuk digunakan pada bahasa pemrograman C/C++, namun dalam perkembangannya OpenGL dapat juga digunakan dalam bahasa pemrograman yang lain seperti Java, Tcl, Ada, Visual Basic, Delphi, maupun Fortran. Namun OpenGL di-*package* secara berbeda-beda sesuai dengan bahasa pemrograman yang digunakan. Oleh karena itu, *package* OpenGL tersebut dapat di-*download* pada situs <http://www.opengl.org> sesuai dengan bahasa pemrograman yang akan digunakan.

Masing-masing perintah atau fungsi dalam OpenGL mempunyai struktur dan format yang sama. Tabel 2.1 menunjukkan beberapa contoh perintah yang biasa digunakan pada OpenGL.

Tabel 2.1 Contoh Perintah-Perintah dalam OpenGL

|  |  |  |
| --- | --- | --- |
| Perintah | Arti | Keterangan |
| glVertex2i(x,y); | Lokasi titik berada di (x,y) | Tipe argumennya adalah integer dan 2 dimensi yaitu x dan y |
| glVertex2f(x,y); | Lokasi titik berada di (x,y) | Tipe argumennya adalah float dan 2 dimensi yaitu x dan y |
| glVertex3i(x,y,z); | Lokasi titik berada di (x,y,z) | Tipe argumennya adalah integer dan 2 dimensi yaitu x, y dan z |
| glVertex3f(x,y,z); | Lokasi titik berada di (x,y,z) | Tipe argumennya adalah float dan 2 dimensi yaitu x, y dan z |
| glClearColour(R, G, B, α); | Warna latar belakang | Empat komponen warna yaitu Red, Green, Blue dan alpha |
| glColor3f(R, G, B); | Warna latar muka (pena) | Tiga komponen warna yaitu Red, Green dan Blue |
| glColor4f(R, G, B); | Warna latar muka (pena) | Empat komponen warna yaitu Red, Green, Blue dan alpha |
| glPointSize(k); | Ukuran titik k piksel | Besar kecilnya ukuran titik tergantung pada k (integer) |
| glBegin(GL\_POINTS); | Titik | Objek primitive (lihat gambar 2.2) |
| glBegin(GL\_LINES); | Garis | Objek primitive (lihat gambar 2.2) |
| glBegin(GL\_LINE\_STRIP); | Poligaris | Objek primitive (lihat gambar 2.2) |
| glBegin(GL\_LINE\_LOOP); | Poligaris tertutup (polygon) | Objek primitive (lihat gambar 2.2) |
| glBegin(GL\_TRIANGLES); | Segitiga | Objek primitive (lihat gambar 2.2) |
| glBegin(GL\_TRIANGLE\_STRIP); | Segitiga | Objek primitive (lihat gambar 2.2) |
| glBegin(GL\_TRIANGLE\_FAN); | Segitiga | Objek primitive (lihat gambar 2.2) |
| glBegin(GL\_QUADS); | Segiempat | Objek primitive (lihat gambar 2.2) |
| glBegin(GL\_QUAD\_STRIP); | Segiempat | Objek primitive (lihat gambar 2.2) |
| glBegin(GL\_POLYGON); | Poligon | Objek primitive (lihat gambar 2.2) |
| glBegin(GL\_LINE\_STIPPLE); | Garis putus-putus | Objek primitive |
| glBegin(GL\_POLY\_STIPPLE); | Poligon dengan pola tertentu | Objek primitive |
| glRect(GLint x1, GLint y1, GLint x2, GLint y2); | Segiempat siku-siku | Objek primitive dan ukuran segiempat ditentukan oleh dua titik yaitu (x1,y1) dan (x2,y2) |
| glEnd( ); | Akhir perintah OpenGL | - |

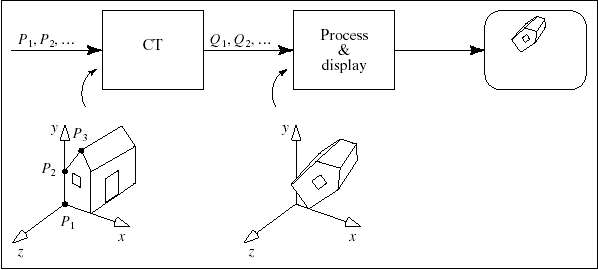


Gambar 1 Contoh-Contoh Objek Primitif

**Gambar 2.2 Contoh-contoh Objek Primitif**

Perintah tranformasi pada OpenGL adalah

* Translasi – **glTranslated**
* Skala – **glScaled**
* Putar atau rotasi – **glRotated**

****

Gambar 2 Transformasi

Tabel 2.2 Format Fungsi OpenGL

|  |  |  |  |
| --- | --- | --- | --- |
| Suffix | Tipe data | C atau C++ | OpenGL |
| B | Integer 8-bit | signed char | GLbyte |
| S | Integer 16-bit | short | GLshort |
| I | Integer 32-bit | int atau long | GLint, GLsizei |
| F | Floating 32-bit | float | GLfloat, GLclampf |
| D | Floating 64-bit | double | GLdouble, GLclampd |
| Ub | unsigned 8-bit | unsigned char | GLubyte, GLboolean |
| Us | unsigned 16-bit | unsigned short | GLushort |
| Ui | unsigned 32-bit | unsigned int atau unsigned long | GLuint, GLenum, GLbitfield |

## Objek 3 Dimensi

Objek tiga dimensi adalah sebuah model struktur data yang menyatakan suatu gambar 3D dibentuk dan disusun. Objek 3Ddidefinisikan dengan :

1. Objek 3D adalah sekumpulan titik-titik 3D (x,y,z) yang membentuk luasan-luasan *(face)* yang digabungkan menjadi satu kesatuan.
2. *Face* adalah gabungan titik-titik yang membentuk luasan tertentu atau sering dinamakan dengan sisi.

Dari definisi ini, dapat dikatakan bahwa objek 3D merupakan kumpulan titik-titik dan kumpulan face yang merupakan susunan dari titik-titik yang ditentukan. Seperti gambar kubus, kubus terdiri dari 8 titik dan 6 sisi/*face*.Dimana *face* merupakan polygon atau kumpulan titik-titik yang disusun urutannya. Dalam kubus, *face-*nya semua berupa bujursangkar dengan 4 titik.

## Bitmap

Bitmap adalah representasi atau gambaran yang terdiri dari baris dan kolom pada titik image graphics di komputer. Nilai dari titik disimpan dalam satu atau lebih data bit. Tampilan dari bitmap atau raster, menggunakan titik-titik berwarna yang dikenal dengan sebutan pixel. Pixel-pixel tersebut ditempatkan pada lokasi-lokasi tertentu dengan nilai-nilai warna tersendiri, yang secara keseluruhan akan membentuk sebuah tampilan gambar. Tampilan bitmap mampu menunjukkan kehalusan gradasi bayangan dan warna dari sebuah gambar, karena itu bitmap merupakan media elelktronik yang paling tepat untuk gambar-gambar dengan perpaduan gradasi warna yang rumit seperti foto dan lukisan digital.

Struktur bitmap terdiri dari Header, Info Header dan Color Tabel. Header adalah bagian dari file bitmap yang berisi informasi header dari file gambar bitmap. Ukuran dari header ini 14 byte, masing-masing terdiri dari signature 2 bytes (berisi “BM” sebagai tanda gambar mempunyai format bmp), FileSize 4 bytes (besarnya ukuran gambar mempunyai satuan bytes), Reserved 4 bytes (tidak digunakan atau sama diisi dengan nilai nol) dan DataOffset 4 bytes (file offset untuk raster data). Info header adalah bagian dari header yang berisi informasi lebih detail dari file gambar bitmap. Color table adalah table yang berisi warna-warna yang ada pada gambar bitmap.

## Lighting (Diffuse, Ambient dan Specular)

Proses menghitung intensitas cahaya terutama pada 3D point, biasanya diatas suatu permukaan :

* Bayangan

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

* Jenis Bayangan

Pada Maya, suatu sumber cahaya bisa tidak menghasilkan bayangan (*default*) atau bisa menghasilkan bayangan *depthmap* maupun *raytraced*. Anda dapat mengombinasikan kedua jenis bayangan ini *depthmap* maupun *raytraced* pada *scene* Anda. Dengan mengatur atribut bayangan *depth map* atau *raytraced*, Anda dapat mensimulasikan Bayangan yang dihasilkan oleh berbagai tipe cahaya di dunia nyata. Bayangan *depth map* maupun *raytraced*memberikan efek yang hamper sama, namun bayangan depth map waktu *render*-nya lebih cepat. Umumnya kebanyakan orang akan memilih bayangan depth map kecuali jika tipe bayangan tersebut tidak dapat membantu mencapai visualisasi yang diinginkan.



Gambar 3 Penambahan bayangan akan meningkatkan kesan realistik

* Model Bayangan dibagi menjadi beberapa bagian :

1. Cahaya *diffuse*(tersebar)

* Perhitungan cahaya tersebar menggunakan m,v dan s.
* Sebagaimana cahaya tersebar disebarkan secara seragam dalam semiua arah, lokasi mata, v, tidak penting kecuali kalau v. m < 0 jika diinginkan intensitas cahaya l = 0
* Hubungan antara kecerahan permukaan dan orientasinya cahaya.



Gambar 4 Contoh *Diffuse*

1. Cahaya *Specular* (Spekular)

* Objek nyata tidak menyebarkan cahaya secara seragam
* Komponen spekular perlu diperhitungkan
* Digunakan model phong, jumlah cahaya dipantulkan terbesar dalam arah pantulan cermin
* Ini merupakan arah semua lintasan cahaya terpantul untuk cermin yang sempurna



Gambar 5 Contoh *Specular*

1. Cahaya *Ambient* (Lingkungan)

* Cahaya jatuh ke objek dari berbagai sudut pantul dari objek lain dan lingkungan
* Secara perhitungan sangat mahal untuk di hitung
* Cahaya ambient tidak mempunyai titik asal khusus.
* Koefesien cahaya ambient : pa’ dipakaikan untuk masing-masing permukaan.
* Sumber intensitas la dikalikan dengan koefesien dan digunakan dan ditambahkan ke sembarang cahaya tersebar atau spekular.

## *Mapping* (Pemetaan)

*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:

* Menentukan tekstur
* Membaca atau membangkitkan tekstur
* Menandai tekstur
* Mengenablekan tekstur
* Menandai koordinat tekstur pada vertex
* Menentukan parameter tekstur
* *wrapping* , *filtering*, dsb.

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

1. Menentukan Tekstur Image

Cara menentukan tekstur image adalah sebagai berikut :

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

* Gambar yang didefinisikan (baik secara manual maupun dengn suatu fungsi matematik tertentu)
* Membangkitkan dengan kode aplikasi

1. Mengenablekan tekstur mapping

* glEnable(GL\_TEXTURE\_2D)
* OpenGL mendukung 1 sampai 4 dimensional tekstur mapping

b. Mendefinisikan gambar sebagai sebuah tekstur

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

Keterangan :

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

c. Mengubah gambar tekstur :

* OpenGL meminta dimensi tekstur untuk menjadi dasar dari 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);
* data\_in adalah gambar inputan.
* data\_out adalah gambar hasil

d. Mapping Tekstur :

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

e. Parameter tekstur

* OpenGL mempunyai berbagai parameter yang menentukanbagaimana tekstur diterapkan
* Parameters wrapping menentukan apa yang terjadi pada s dan t
* diluar jangkauan (0,1)
* Mode filter mengijinkan penggunaan area rata-rata sebagai ganti contoh titik
* Mipmapping mengijinkan penggunaan tekstur pada berbagai resolusi
* Parameter lingkungan menentukan bagaimana tekstur mapping berinteraksi dengan shading

f. Magnification dan Minification

* Lebih dari satu texel dapat dicakup sebuah pixel (minification) atau lebih dari satu piksel dapat dicakup sebuah texel (magnification). Dapat menggunakan contoh titik (texel terdekat ) atau filter linier (filter 2x2) untuk mengisi nilai tekstur
* Modes ditentukan oleh :
* glTexParameteri(target, type, mode )
* glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXURE\_MAG\_FILTER,GL\_NEAREST);
* glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXURE\_MIN\_FILTER,GL\_LINEAR);
* Catatan bahwa filter linier tersebut meminta sebuah batas dari extra texel untuk filter pada tepi (batas = 1 )

## *Perspective*

*Perspective* adalah sudut-sudut diantara garis yang membuat kenampakan dari ilusi 3 dimensi. Gambar 6 berikut merupakan gambar kubus 3 dimensi yang disusun dari garis-garis. Gambar tersebut telah mengilustrasikan sebuah kubus, tetapi masih menimbulkan persepsi yang membingungkan. Sedangkan Gambar 7 memberikan ilusi 3 dimensi yang lebih kuat dengan menghilangkan garis yang tertutup (*hidden line remofal*) dan menghapus bidang atau sisi belakang yang tak nampak (*hidden surface remofal*) untuk menghasilkan kenampakan permukaan yang solid.

|  |  |
| --- | --- |
| Gambar 6Kubus 3 Dimensi yang tersusun dari Garis | Gambar 7Image Kubus yang lebih baik |

## *Color* and *Shading*

Untuk memberikan kesan bidang 3 dimensi yang lebih baik kita dapat memberikan pewarnaan terhadap objek tersebut. Tetapi teknik pewarnaan tersebut harus digunakan dengan hati-hati untuk menghasilkan efek diinginkan. Jika kita mewarnai kubus dengan suatu warna yang sama justru akan menghilangkan kesan 3 dimensi dari kubus tersebut seperti pada gambar 8 (seluruh bidang diberi warna yang sama). Sedangkan teknik pewarnaan yang baik akan menambah kesan ilusi 3 dimensi seperti pada gambar 9 (setiap bidang diberi warna yang berbeda). Sama halnya dengan pemberian shading seperti pada gambar 10 akan memberikan kenampakan/ilustrasi objek 3 dimensi yang lebih kuat.

|  |  |
| --- | --- |
| Gambar 8 Penambahan warna yang menimbulkan kebingungan | Gambar 9 Menambahkan warma yang berbeda akan menimbulkan ilusi 3 dimensi yang lebih baik |



Gambar 10 Pemberian Shading yang cukup akan meningkatkan ilusi obek 3 dimensi

## *Blending*(Pencampuran)

Pencampuran merupakan fungsi yang menggabungkan nilai warna dari sumber dan tujuan. Pencampuran dapat mengontrol berapa banyak warna yang dapat dikombinasikan. Dengan demikian untuk proses membuat terang fragmen dapat menggunakan alpha pencampuran. Warna pencampuran terletak pada teknik utama, seperti transparan, digital composite dan lukisan. 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. Dengan demikian, transparan mempunyai permukaan lebih rendah daripada yang buram.

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)*.Teknik *blending* juga digunakan untuk ilusi dari cerminan sebuah objek pada gambar 11.



Gambar 11 Tehnik *blending* untuk mendapatkan efek pantulan

## *Antialiasing*

*Antialiasing* adalah efek yang muncul akibat fakta bahwa image yang dibuat tersusun oleh pixel-pixel yang bersifat diskrit. Seperti pada gambar 12, terlihat bahwa garis yang membentuk kubus tersebut memiliki *jagged edges* (*jaggies*). Dengan melakukan *blending* pada garis terhadap warna latar, kita dapat menghilangkan *jaggies* dan menghasilkan tampilan garis yang mulus (*smooth*). Teknik tersebut dinamai teknik *Antialiasing*.



Gambar 12*Jagged Lines VS Smooth Lines*

## *Translation*

Sebuah objek dalam tiga dimensi di translasikan dengan mengubah setiap setiao point yang mendefinisikan objek. Untuk sebuah objek yang dibangun atau direpresentasikan dengan sebuah set dari permukaan – permukaan poligon , translasi dilakukan pada setiap permukaan dan menggambar kembali permukaan poligon pada posisi baru.



Gambar 13Translasi Objek 3D

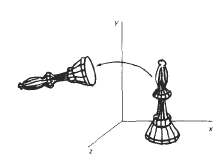
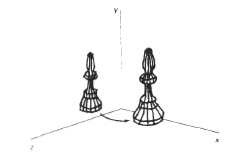
## *Rotation*

Untuk membangkitkan rotasi pada objek 3D kita harus membuat aksis dari rotasi dan jumlah sudut rotasi . Tidak seperti melakukan rotasi pada objek 2D yang semua proses transformasi dilakukan di koordinat xy , sebuah rotasi objek tiga dimensi bisa dilakukan di *space* manapun.

Dengan menggunakan notasi matrix, maka besaran R bisa dikatakan sbb:

**R = cos(0) sin(0)**

**-sin(0) cos(0)**

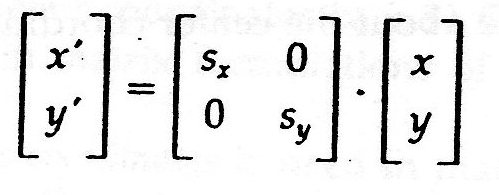


Gambar 14 Rotasi objek 3D pada Sumbu x , y , dan z

## *Scaling*

Penskalaan pada objek 3D dengan transformasi mengubah ukuran dan posisi objek relatif terhadap koordinat asli. Jika tidak semua parameternya sama dimensi – dimensi relative pada objek akan diubah.

Matriks transformasi untuk skala dapat juga dinyatakn sebagai berikut :



|  |  |
| --- | --- |
| Gambar 15 Proses Penskalaan Objek 3 Dimensi |  |

# ANALISIS DAN PERANCANGAN

## Analisis Sistem

Sistem yang akan dibangun adalah sebuah aplikasi yang menghasilkan efek – efek dari*lighting, color, tranformasi, antialiasing, diffuse, ambient, specular, blending* dan *mapping*. Pada bab I telah dijelaskan mengenai batasan masalah di dalam pengimplementasian aplikasi yang dimaksud.

Karena proses penggambaran dilakukan oleh hadware, maka dibutuhkan bahasa pemrograman yaitu menggunakan bahas pemrograman visual C++. Adapun kebutuhan software dan hadwarenya akan dijelaskan lebih lanjut.

### Analisis Kebutuhan Non-Fungsional

Analisis kebutuhan non fungsional dilakukan untuk menghasilkan spesifikasi kebutuhan non fungsional. Spesifikasi kebutuhan non fungsional adalah spesifikasi yang rinci tentang hal-hal yang dibutuhkan oleh system sebelum system diimplementasikan.

#### Analisis *Hardware*

Space untuk menjalankan aplikasi ini kami merekomendasikan kebutuhan hardware sebagai berikut :

* Prosesor 1.60 Ghz
* Ram 1 Gb
* Keyboard dan Mouse
* VGA 128MB

#### Anaisis Software

Dan untuk software kami menggunakan software sebagai berikut :

* Sistem Operasi (Microsoft Windows XP Home)
* Miscosoft Visual C ++ v.6.0
* Glut32.dll

### Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional dilakukan untuk menghasilkan spesifikasi kebutuhan fungsional. Spesifikasi kebutuhan fungsional adalah spesifikasi yang rinci tentang hal-hal yang akan dilakukan sistem ketika diimplementasikan.

Perangkat lunak yang dibuat akan menangani satu fungsi utama, seperti terlihat dalam use case berikut:



Gambar 16 Tampilan Aplikasi

#### Tampilan Aplikasi

Fungsi ini digunakan untuk menjalankan aplikasipembuatan objek 3D yaitu berupa objek di luar angkasa.

#### Skenario

|  |  |  |
| --- | --- | --- |
| **Identifikasi** | | |
| **Nama** | Tampilan. | |
| **Tujuan** | Mendeskripsikan aplikasi. | |
| **Deskripsi** | Menjelaskan fungsi aplikasi yang dibangun. | |
| **Tipe** |  | |
| **Aktor** | User | |
| **Skenario Utama** | | |
| **Kondisi awal** | Tampilan aplikasi | |
| **Aksi Aktor** | | **Reaksi Sistem** |
| Menjalankan dan melihat aplikasi. | | Membaca dan memproses tampilan aplikasi. |
| **Kondisi akhir** | Menampilkan tampilan aplikasi, keluar aplikasi, atau layar berubah. | |

#### Dinamika Objek



Gambar 17 Dinamika Objek Aplikasi

#### Tampilan Aplikasi Posisi Cahaya



#### Skenario

|  |  |  |
| --- | --- | --- |
| **Identifikasi** | | |
| **Nama** | Tampilan Aplikasi Posisi Cahaya | |
| **Tujuan** | Mendeskripsikan AplikasiPosisi Cahaya | |
| **Deskripsi** | Menjelaskan fungsi aplikasi yang dibangun. | |
| **Tipe** |  | |
| **Aktor** | User | |
| **Skenario Utama** | | |
| **Kondisi awal** | Tampilan aplikasi dengan objek diam | |
| **Aksi Aktor** | | **Reaksi Sistem** |
| Menjalankan dan melihat aplikasi. | | Membaca dan memproses tampilan aplikasi. |
| **Kondisi akhir** | Menampilkan tampilan aplikasi yang menentukan posisi cahaya. | |

#### Dinamika Objek



Gambar 18 Dinamika Objek Posisi Cahaya

#### Tampilan Aplikasi Posisi Objek Satelit



#### Skenario

|  |  |  |
| --- | --- | --- |
| **Identifikasi** | | |
| **Nama** | Tampilan Aplikasi Posisi Objek Satelit. | |
| **Tujuan** | Mendeskripsikan aplikasi Aplikasi Posisi Satelit | |
| **Deskripsi** | Menjelaskan fungsi aplikasi yang dibangun. | |
| **Tipe** |  | |
| **Aktor** | User | |
| **Skenario Utama** | | |
| **Kondisi awal** | Tampilan aplikasi dengan objek satelit berotasi | |
| **Aksi Aktor** | | **Reaksi Sistem** |
| Menjalankan dan melihat aplikasi. | | Membaca dan memproses tampilan aplikasi. |
| **Kondisi akhir** | Menampilkan tampilan aplikasi yang menentukan posisi objek Satelit. | |

#### Dinamika Objek



Gambar 19 Dinamika Objek Posisi Satelit

#### Tampilan Aplikasi Posisi Objek Bumi dan Bulan



#### Skenario

|  |  |  |
| --- | --- | --- |
| **Identifikasi** | | |
| **Nama** | Tampilan Aplikasi Posisi Objek Bumi dan Bulan. | |
| **Tujuan** | Mendeskripsikan aplikasi Posisi Bumi dan Bulan | |
| **Deskripsi** | Menjelaskan fungsi aplikasi yang dibangun. | |
| **Tipe** |  | |
| **Aktor** | User | |
| **Skenario Utama** | | |
| **Kondisi awal** | Tampilan aplikasi dengan objek Bumi dan Bulan berotasi | |
| **Aksi Aktor** | | **Reaksi Sistem** |
| Menjalankan dan melihat aplikasi. | | Membaca dan memproses tampilan aplikasi. |
| **Kondisi akhir** | Menampilkan tampilan aplikasi yang menentukan posisi objek Bumi dan Bulan. | |

#### Dinamika Objek



Gambar 20 Dinamika Objek Posisi Bumi dan Bulan

## Perancangan

Tahap implementasi sistem merupakan tahap menterjemahkan perancangan dalam bahasa yang dapat dimengerti oleh mesin serta penerapan perangkat lunak pada keadaan yang sesungguhnya. Dan merupakan tahap penerjemahan kebutuhan pembuatan aplikasi ke dalam representassi perangkat lunak.

### Perancangan *Interface*(Antarmuka)

Perancangan *Interface* merupakan suatu bentuk tampilan dari aplikasi yang akan dibuat untuk kebutuhan *interface* dengan user**.**  Berikut ini merupakan rancangan Aplikasi yang akan dibangun :



**Gambar 16 Perancangan Antarmuka**

### Perancangan Perintah Melalui Keyboard

Perancangan perintah yang akan dilakukan pada aplikasi yang akan dibangun yaitu dengan menggunakan keyboard. Berikut perintah yang akan diterapkan :

Tabel 2.3 Perintah pada Keyboard

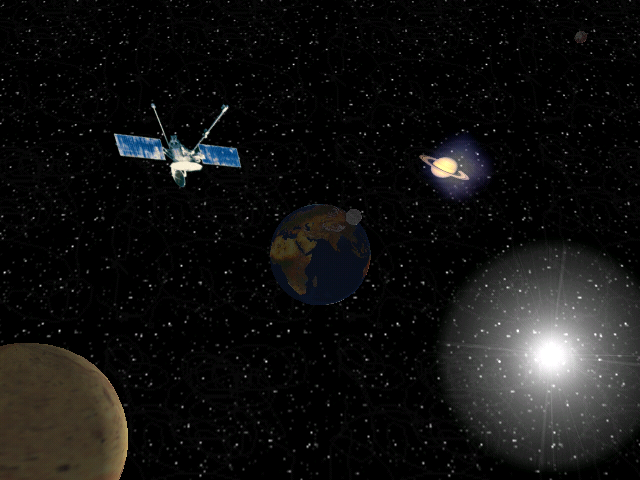
|  |  |  |
| --- | --- | --- |
| 1 | Key-Q | : Menggerakan satelit ke dalam |
| 2 | Key-E | : Menggerakan satelit ke keluar |
| 3 | Key-W | : Menggerakan satelit ke atas |
| 4 | Key-S | : Menggerakan satelit ke bawah |
| 5 | Key-A | : Menggerakan satelit ke kiri |
| 6 | Key-D | : Menggerakan satelit ke kanan |
| 7 | Key-I | : Menggerakan cahaya ke atas |
| 8 | Key-J | : Menggerakkan cahayai ke kiri |
| 9 | Key-K | : Menggerakkan cahayake bawah |
| 10 | Key-L | : Menggerakkan cahaya ke kanan |
| 11 | Key-U | : Menggerakkan cahaya ke dalam |
| 12 | Key-O | : Menggerakkan cahaya ke luar |
| 13 | Key-up | : Menggerakkan bumi dan bulan ke atas |
| 14 | Key-down | : Menggerakkan bumi dan bulan ke bawah |
| 15 | Key-left | : Menggerakkan bumi dan bulan ke kiri |
| 16 | Key-right | : Menggerakkan bumi dan bulan ke kanan |
| 17 | Key-Z | : Menggerakkan bumi dan bulan ke dalam |
| 18 | Key-X | : Menggerakkan bumi dan bulan ke luar |
| 19 | F1 | : Merubah layar tampilan (*Fullscreen* atau tidak) |
| 20 | ESC | : Keluar aplikasi |

# IMPLEMENTASI

Tahap implementasi sistem merupakan tahap menterjemahkan perancangan berdasarkan hasil analisis dalam bahasa yang dapat dimengerti oleh mesin serta penerapan perangkat lunak pada keadaan yang sesungguhnya.

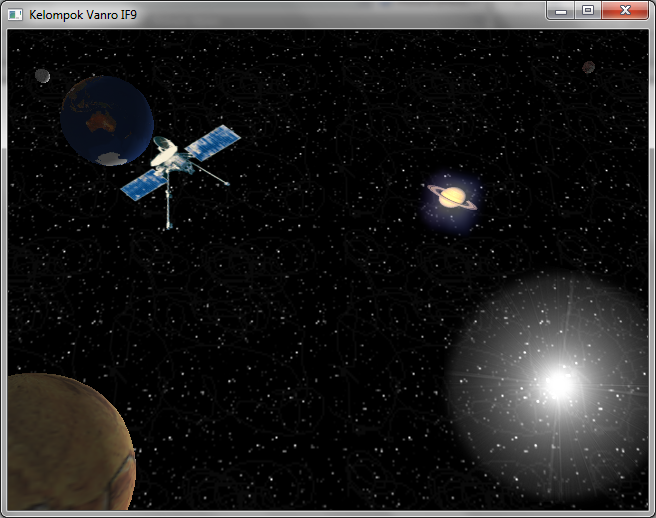
## Tampilan Awal

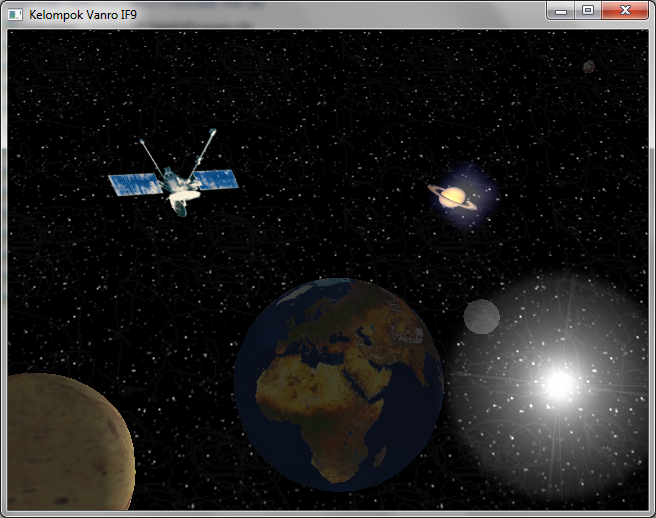
Berikut ini merupakan tampilan antar muka aplikasi 3 Dimensi menggunakan objek Luar Angkasa dengan menggunakan Visual C++. Tampilan awal dari aplikasi dapat dilihat pada gambar berikut ini.

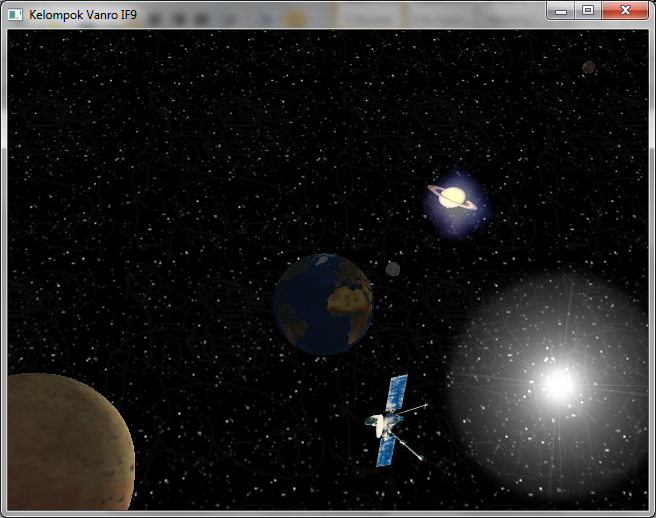


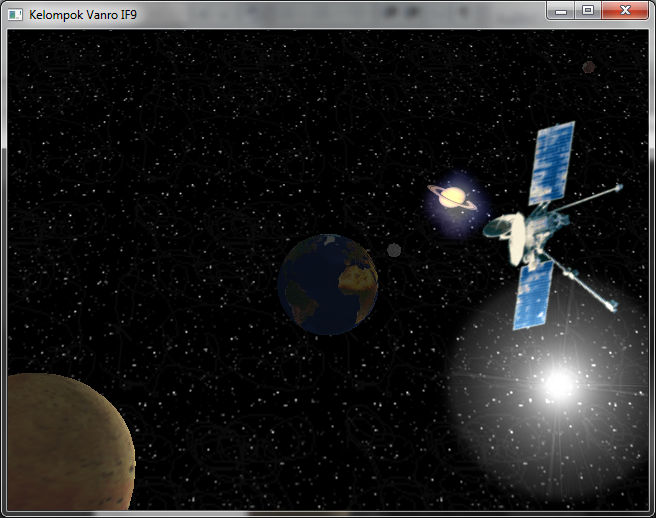
Gambar 17 Tampilan Awal Aplikasi

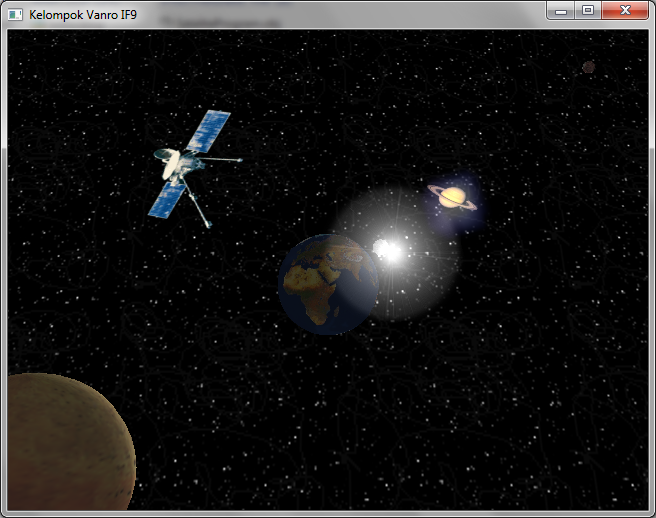
## Tampilan dari beberapa *capture*

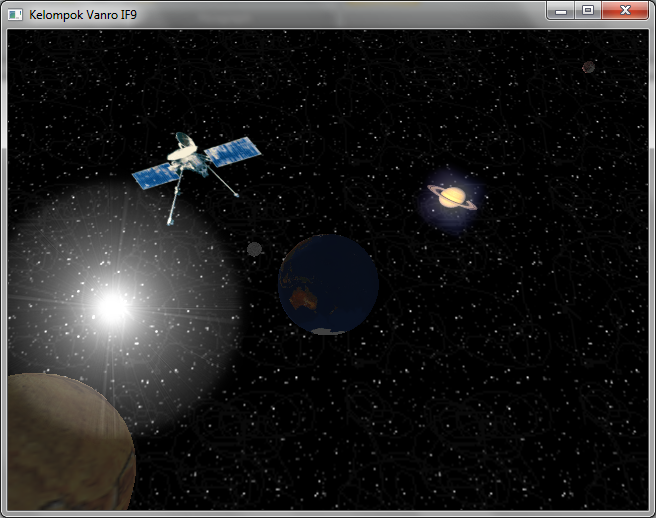
1. Menggerakkan Bumi



1. Menggerakkan Satelit



1. Menggerakkan Cahaya



# KESIMPULAN

Kesimpulan dari aplikasi yang dibangun adalah :

1. Grafika Komputer adalah suatu proses pembuatan, penyimpanan dan manipulasi model, citra atau gambar berdasarkan deskripsi obyek maupun latar belakang yang terkandung pada gambar tersebut. Dimana model berasal dari beberapa bidang seperti fisik, matematik, artistik dan bahkan struktur abstrak.
2. OpenGL adalah sebuah antarmuka pemograman yang digunakan dalam membuat grafik real-time 3 dimensi.
3. Teknik animasi bertujuan memberikan kesan nyata (realisme) pada objek dengan mengaplikasikan teknik-teknik 3D seperti : *perspektif, color and shading, lightingand shadows*, *antialiasing* pada objek dan *mapping texture*.

# DAFTAR PUSKATA

http://vicknite.wordpress.com/2010/05/25/basic-of-opengl-3d-project-tutorial/

http://hoirul.itssby.edu/Grafikakomputer/bahan%20bacaaan/NeHe%20Productions\_%20OpenGL%20Lesson%20%2311.pdf

http://delpanantika.com/search/OPENGL+WITH+VC++

http://www.falloutsoftware.com/tutorials/gl/gl7.htm

Suyoto.2003.Teori dan Pemograman Grafika Komputer dengan Visual C++ V.6 dan OpenGL. Yogyakarta. Grava Media.

# LAMPIRAN

|  |
| --- |
| #include <windows.h>  #include <gl\gl.h>  #include <gl\glu.h>  #include <gl\glaux.h>  #include <stdio.h>  #include <math.h>  HDC hDC=NULL;  HGLRC hRC=NULL;  HWND hWnd=NULL;  HINSTANCE hInstance;  bool keys[256];  bool active=TRUE;  bool fullscreen=TRUE;  ////////////////////Inisilaisasi Global//////////////////////////////////////  float SatelitePos[] = {-0.7f,0.5f,-3.0f};  float EarthPos[] = {0.0f,1.3f, -4.0f};  float LightPos[] = {0.4f, -0.2f, -1};  GLint base,loop;  GLuint texture[14];  GLUquadricObj \*Obiekt1;  GLfloat roll,roll2;  GLfloat step=0.01f;  GLfloat step2=0.3f;  GLfloat rotate, rotateobiekt1,rotateobiekt2;  GLfloat angle,rotsat;  GLfloat RotateWenus;  GLfloat x,y;  GLfloat LightAmbient[]= { 0.05f, 0.05f, 0.05f, 1.0f };  GLfloat LightDiffuse[]= { 0.8f, 0.8f, 0.8f, 1.0f };  GLfloat LightPosition[]= { 10.0f, -10.0f, 10.0f, 1.0f };  GLfloat roo;  GLfloat ro;  GLfloat ro2;  GLfloat alpha;  GLfloat alphaplus=0.0003f;  int Tekst = 0;  int TextSync = 0;  ////////////////////////////////////////////////////////////////////////////////////////  LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);  void Build\_font(void)  {  int i;  float cx; // Holds Our X Character Coord  float cy; // Holds Our Y Character Coord  base = glGenLists(256); // Creating 256 Display Lists  glBindTexture(GL\_TEXTURE\_2D, texture[4]); // Select Our Font Texture  for (i=0 ; i<256 ; i++) { // Loop Through All 256 Lists  cx = float(i%16)/16.0f; // X Position Of Current Character  cy = float(i/16)/16.0f; // Y Position Of Current Character  glNewList(base+i,GL\_COMPILE); // Start Building A List  glBegin(GL\_QUADS); // Use A Quad For Each Character  glTexCoord2f(cx,1-cy-0.0625f); // Texture Coord (Bottom Left)  glVertex2i(0,0); // Vertex Coord (Bottom Left)  glTexCoord2f(cx+0.0625f,1-cy-0.0625f); // Texture Coord (Bottom Right)  glVertex2i(16,0); // Vertex Coord (Bottom Right)  glTexCoord2f(cx+0.0625f,1-cy); // Texture Coord (Top Right)  glVertex2i(16,16); // Vertex Coord (Top Right)  glTexCoord2f(cx,1-cy); // Texture Coord (Top Left)  glVertex2i(0,16); // Vertex Coord (Top Left)  glEnd(); // Done Building Our Quad (Character)  glTranslated(10,0,0); // Move To The Right Of The Character  glEndList(); // Done Building The Display List  } // Loop Until All 256 Are Built  }  // Delete The Font From Memory (NEW)  void KillFont(void)  {  glDeleteLists(base, 256); // Delete All 256 Display Lists  }  void glPrint(GLint x, GLint y, char\* string, int set)  {  if (set > 1) set = 1;  glBindTexture(GL\_TEXTURE\_2D, texture[4]); // Select Our Font Texture  glDisable(GL\_DEPTH\_TEST); // Disables Depth Testing  glMatrixMode(GL\_PROJECTION); // Select The Projection Matrix  glPushMatrix(); // Store The Projection Matrix  glLoadIdentity(); // Reset The Projection Matrix  glOrtho(0,640,0,480,-100,100); // Set Up An Ortho Screen  glMatrixMode(GL\_MODELVIEW); // Select The Modelview Matrix  glPushMatrix(); // Store The Modelview Matrix  glLoadIdentity(); // Reset The Modelview Matrix  glTranslated(x,y,0); // Position The Text (0,0 - Bottom Left)  glListBase(base-32+(128\*set)); // Choose The Font Set (0 or 1)  glCallLists(strlen(string),GL\_BYTE,string); // Write The Text To The Screen  glMatrixMode(GL\_PROJECTION); // Select The Projection Matrix  glPopMatrix(); // Restore The Old Projection Matrix  glMatrixMode(GL\_MODELVIEW); // Select The Modelview Matrix  glPopMatrix(); // Restore The Old Projection Matrix  glEnable(GL\_DEPTH\_TEST); // Enables Depth Testing  }  ///////////////////////Mengeset Pencahayaan/////////////////////////////////////////  void Swiatlo(void)  {  glLightfv(GL\_LIGHT1, GL\_AMBIENT, LightAmbient); // Mengeset Cahaya ambient  glLightfv(GL\_LIGHT1, GL\_DIFFUSE, LightDiffuse); // Mengeset Cahaya diffuse  glLightfv(GL\_LIGHT1, GL\_POSITION,LightPosition); // Posisi cahaya  glEnable(GL\_LIGHT0);  glEnable(GL\_LIGHT1); // Mengaktifkan Cahaya satu  glEnable(GL\_LIGHT2);  glEnable(GL\_LIGHTING);  glEnable(GL\_COLOR\_MATERIAL);  }  ///////////////////////////////////Meload Gambar bertipe Bitmap////////////////////////////////  AUX\_RGBImageRec \*LoadBMP(char \*Filename)  {  FILE \*File=NULL;  if (!Filename)  {  return NULL;  }  File=fopen(Filename,"r");  if (File)  {  fclose(File);  return auxDIBImageLoad(Filename);  }  return NULL;  }  ////////////////////////////////////////Prosedur membuat tekstur//////////////////////////////////////  int LoadTextures()  {  int Status = FALSE;  AUX\_RGBImageRec \*TextureImage[14];  memset(TextureImage, 0, sizeof(void \*)\*14);  if ( (TextureImage[0]=LoadBMP("kosmos.bmp")) &&  (TextureImage[1]=LoadBMP("belka.bmp")) &&  (TextureImage[2]=LoadBMP("ziemia.bmp"))&&  (TextureImage[3]=LoadBMP("moon.bmp"))&&  (TextureImage[4]=LoadBMP("fonty.bmp")) &&  (TextureImage[5]=LoadBMP("mars.bmp")) &&  (TextureImage[6]=LoadBMP("wenus.bmp")) &&  (TextureImage[7]=LoadBMP("merkury.bmp")) &&  (TextureImage[8]=LoadBMP("satelitemaska.bmp")) &&  (TextureImage[9]=LoadBMP("satelite.bmp")) &&  (TextureImage[10]=LoadBMP("saturnmaska.bmp")) &&  (TextureImage[11]=LoadBMP("saturn.bmp")) &&  (TextureImage[12]=LoadBMP("space.bmp")) &&  (TextureImage[13]=LoadBMP("sun.bmp")))  {  Status=TRUE;  glGenTextures(14, &texture[0]);  for(loop=0;loop<14;loop++)  {  glBindTexture(GL\_TEXTURE\_2D, texture[loop]);  gluBuild2DMipmaps(GL\_TEXTURE\_2D, 3,TextureImage[loop]->sizeX, TextureImage[loop]->sizeY, GL\_RGB, GL\_UNSIGNED\_BYTE, TextureImage[loop]->data);  glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR); // Linear Filtering  glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR); // Linear Filtering  }  for(loop=0;loop<14;loop++)  {  if (TextureImage[loop])  {  if (TextureImage[loop]->data)  free(TextureImage[loop]->data);  free(TextureImage[loop]);  }  }  }  return Status;  }  ////////////////////////////////Mengubah ukuran tampilanwindow//////////////////////////////////////  GLvoid ReSizeGLScene(GLsizei width, GLsizei height)  {  if (height==0)  {  height=1;  }  glViewport(0,0,width,height);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,100.0f);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  }  //////////////////////////////////Prosuder main OpenGL/////////////////////////////////////////  int InitGL(GLvoid)  {  if(!LoadTextures())  return FALSE;  Swiatlo();  glClearColor(0.0f, 0.0f, 0.0f, 0.5f); // Background warna Hitam  glClearDepth(1.0f);  glEnable(GL\_DEPTH\_TEST);  glShadeModel(GL\_SMOOTH);  glEnable(GL\_TEXTURE\_2D);  glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST);  glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST);  Obiekt1=gluNewQuadric();  gluQuadricTexture(Obiekt1, GLU\_TRUE);  gluQuadricDrawStyle(Obiekt1, GLU\_FILL);  return TRUE;  }  GLint DrawGLScene(GLvoid) // Here's Where We Do All The Drawing  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear Screen And Depth Buffer  glLoadIdentity();  gluLookAt(0,0,10, 0,0,0, 0,1,0);  glLightfv(GL\_LIGHT1, GL\_POSITION, LightPos);  //MEMBUAT BACKGROUND  glDisable(GL\_LIGHTING);  glTranslatef(0,0,1);  glRotatef(20,-1,0,0);  glRotatef(rotate,0,0,0);  glRotatef(rotate, 0,0,0);    glBindTexture(GL\_TEXTURE\_2D, texture[0]);  glBegin(GL\_QUADS);  glTexCoord2f(0.0f, roll + 0.0f); glVertex3f(-10.1f, -10.1f, 0.0);  glTexCoord2f(6.0f, roll + 0.0f); glVertex3f(10.1f, -10.1f, 0.0);  glTexCoord2f(6.0f, roll + 6.0f); glVertex3f(10.1f, 10.1f, 0.0);  glTexCoord2f(0.0f, roll + 6.0f); glVertex3f(-10.1f, 10.1f, 0.0);  glEnd();    glEnable(GL\_LIGHTING);  //AKHIR MEMBUAT BACKGROUND  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  //MEMBUAT PLANET MARS  glLoadIdentity();  glBindTexture(GL\_TEXTURE\_2D, texture[5]);  glTranslatef(-2.6f,-1.8f,-5);  glRotatef(90,1,0,0);  glRotatef(rotateobiekt2,0,0,1);  gluSphere(Obiekt1, 0.9, 20,20);  //AKHIE MEMBUAT PLANET MARS  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  //MEMBUAT PLANET MERKURY  glLoadIdentity();  glTranslatef(4.5,3.5,-10);  glBindTexture(GL\_TEXTURE\_2D, texture[7]);  glRotatef(rotateobiekt2,1,1,0);  gluSphere(Obiekt1, 0.1,20,20);  //AKHIR MEMBUAT MERKURY  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  //MEMBUAT PLANET BUMI DAN BULAN  glLoadIdentity();  glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE);  glEnable(GL\_BLEND);  glBindTexture(GL\_TEXTURE\_2D, texture[1]);  glTranslatef(0,0,-4);  glTranslatef(0,-1.5,0);  glDisable(GL\_LIGHT1);  glEnable(GL\_LIGHT0);  glDisable(GL\_LIGHT0);  glEnable(GL\_LIGHT1);    glDisable(GL\_BLEND);  glBindTexture(GL\_TEXTURE\_2D, texture[2]);  glTranslatef(EarthPos[0], EarthPos[1], EarthPos[2]); // Merubah Posisi Bumi  glRotatef(rotateobiekt1,0,1,0);  glRotatef(-75,1,0,0);  gluSphere(Obiekt1,0.7,20,20);  glTranslatef(1,0,0.5);  glBindTexture(GL\_TEXTURE\_2D, texture[3]);  glRotatef(rotateobiekt1,1,1,1);  gluSphere(Obiekt1,0.1,20,20);  //AKHIR MEMBUAT PLANET DAN BUMI  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  //MEMBUAT PLANET SATURNUS  glLoadIdentity();  glDisable(GL\_LIGHTING);  glTranslatef(0.60f, 0.35f,-2.8f);  glRotatef(20,0,0,1);  glDisable(GL\_DEPTH\_TEST);  glEnable(GL\_BLEND);  glBlendFunc(GL\_DST\_COLOR, GL\_ZERO);  glBindTexture(GL\_TEXTURE\_2D, texture[10]);  glScalef(0.1f,0.1f,0.1f);  glBegin(GL\_QUADS);  glTexCoord2f(0.0 , 0.0); glVertex3f(-1.0, -1.0, 0.0);  glTexCoord2f(1.0 , 0.0); glVertex3f(1.0, -1.0, 0.0);  glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 0.0);  glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, 0.0);  glEnd();  glBlendFunc(GL\_ONE, GL\_ONE);  glBindTexture(GL\_TEXTURE\_2D, texture[11]);  glBegin(GL\_QUADS);  glTexCoord2f(0.0 , 0.0); glVertex3f(-1.0, -1.0, 0.0);  glTexCoord2f(1.0 , 0.0); glVertex3f(1.0, -1.0, 0.0);  glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 0.0);  glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, 0.0);  glEnd();  glEnable(GL\_DEPTH\_TEST);  glDisable(GL\_BLEND);  glEnable(GL\_LIGHTING);  //AKHIR MEMBUAT PLANET SATURNUS  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  //MENGGAMBAR SATELITE  glLoadIdentity();  glDisable(GL\_LIGHTING);  glTranslatef(SatelitePos[0], SatelitePos[1], SatelitePos[2]); // Position The Satelite  glRotatef(rotsat,0,0,1);  glDisable(GL\_DEPTH\_TEST);  glEnable(GL\_BLEND);  glBlendFunc(GL\_DST\_COLOR, GL\_ZERO);  glBindTexture(GL\_TEXTURE\_2D, texture[8]);  glScalef(0.3f,0.3f,0.3f);  glBegin(GL\_QUADS);  glTexCoord2f(0.0 , 0.0); glVertex3f(-1.0, -1.0, 0.0);  glTexCoord2f(1.0 , 0.0); glVertex3f(1.0, -1.0, 0.0);  glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 0.0);  glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, 0.0);  glEnd();  glBlendFunc(GL\_ONE, GL\_ONE);  glBindTexture(GL\_TEXTURE\_2D, texture[9]);  glBegin(GL\_QUADS);  glTexCoord2f(0.0 , 0.0); glVertex3f(-1.0, -1.0, 0.0);  glTexCoord2f(1.0 , 0.0); glVertex3f(1.0, -1.0, 0.0);  glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 0.0);  glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, 0.0);  glEnd();  glEnable(GL\_DEPTH\_TEST);  glDisable(GL\_BLEND);  glEnable(GL\_LIGHTING);  //AKHIR MENGGAMBAR SATELITE  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//    //ONEMORE  glLoadIdentity();    glTranslatef(1.8f, 1.0f, -8);    glDisable(GL\_LIGHTING);  glEnable(GL\_BLEND);  glColor4f(1,1,1, alpha);  glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE);  glRotatef(ro2,0,0,1);  glBindTexture(GL\_TEXTURE\_2D, texture[12]);  glBegin(GL\_QUADS);  glTexCoord2f(0.0 , 0.0); glVertex3f(-1.0, -1.0, 0.0);  glTexCoord2f(1.0 , 0.0); glVertex3f(1.0, -1.0, 0.0);  glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 0.0);  glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, 0.0);  glEnd();    glRotatef(ro,0,0,1);  glBegin(GL\_QUADS);  glTexCoord2f(0.0 , 0.0); glVertex3f(-1.0, -1.0, 0.0);  glTexCoord2f(1.0 , 0.0); glVertex3f(1.0, -1.0, 0.0);  glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 0.0);  glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, 0.0);  glEnd();  glDisable(GL\_BLEND);  glColor4f(1,1,1,1);  //THAT'S IT    alpha+=alphaplus;  if(alpha>0.5)  alphaplus=-alphaplus;  if(alpha<0.0)  alphaplus=-alphaplus;  ro2+=0.1f;  ro+=0.05f;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  glLoadIdentity();    glTranslatef(LightPos[0], LightPos[1], LightPos[2]); // Position The Sphere  glScalef(0.2f, 0.2f, 0.2f);  glDisable(GL\_LIGHTING);  glEnable(GL\_BLEND);    glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE);  glColor4f(1.1f, 1.99f, 1.4f, 1.5f); // Warna cahaya    glBindTexture(GL\_TEXTURE\_2D, texture[13]);  glBegin(GL\_QUADS);  glTexCoord2f(0.0 , 0.0); glVertex3f(-1.0, -1.0, 0.0);  glTexCoord2f(1.0 , 0.0); glVertex3f(1.0, -1.0, 0.0);  glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 0.0);  glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, 0.0);  glEnd();    glRotatef(roo,0,0,1);    glBindTexture(GL\_TEXTURE\_2D, texture[13]);  glBegin(GL\_QUADS);  glTexCoord2f(0.0 , 0.0); glVertex3f(-1.0, -1.0, 0.0);  glTexCoord2f(1.0 , 0.0); glVertex3f(1.0, -1.0, 0.0);  glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 0.0);  glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, 0.0);  glEnd();  glColor4f(1,1,1,1);  glDisable(GL\_BLEND);    //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  roo+=0.1f;  if(roo>360)  roo=0.0f;  rotsat+=0.1f;  if(rotsat>360)  rotsat=0;  RotateWenus+=0.2f;  if(RotateWenus>360)  RotateWenus=0;  TextSync++;  if (TextSync > 600)  {  TextSync = 0;  Tekst++;  }  if (Tekst > 6)  Tekst = 0;  rotateobiekt1+=step2;  rotateobiekt2+=0.05f;  roll+=0.002f;  if(roll>1.0f)  roll-=1.0f;    roll2+=0.0005f;  if(roll2>1.0f)  roll2-=1.0f;  return TRUE; // Everything Went OK  }  ///////////////////////////////////////////Proses dari ////////////////////////////////////////  void ProcessKeyboard()  {  // Posisi Matahari  if (keys['L']) LightPos[0] += 0.01f; // Kanan  if (keys['J']) LightPos[0] -= 0.01f; // Kiri  if (keys['I']) LightPos[1] += 0.01f; // Atas  if (keys['K']) LightPos[1] -= 0.01f; // Bawah  if (keys['O']) LightPos[2] += 0.01f; // Depan  if (keys['U']) LightPos[2] -= 0.01f; // Belakang  // Posisi Bumi  if (keys[VK\_RIGHT]) EarthPos[0] += 0.01f; // Kanan  if (keys[VK\_LEFT]) EarthPos[0] -= 0.01f; // Kiri  if (keys[VK\_UP]) EarthPos[1] += 0.01f; // Atas  if (keys[VK\_DOWN]) EarthPos[1] -= 0.01f; // Bawah  if (keys['X']) EarthPos[2] += 0.01f; // Depan  if (keys['Z']) EarthPos[2] -= 0.01f; // Belakang  // Posisi Satelite  if (keys['D']) SatelitePos[0] += 0.01f; // Kanan  if (keys['A']) SatelitePos[0] -= 0.01f; // Kiri  if (keys['W']) SatelitePos[1] += 0.01f; // Atas  if (keys['S']) SatelitePos[1] -= 0.01f; // Bawah  if (keys['E']) SatelitePos[2] += 0.01f; // Depan  if (keys['Q']) SatelitePos[2] -= 0.01f; // Belakang  }  ///////////////////////////Menghancurkan Tampilan Window/////////////////////////////////  GLvoid KillGLWindow(GLvoid) // Properly Kill The Window  {  gluDeleteQuadric(Obiekt1);  if (fullscreen) // Are We In Fullscreen Mode?  {  ChangeDisplaySettings(NULL,0); // If So Switch Back To The Desktop  ShowCursor(TRUE); // Show Mouse Pointer  }  if (hRC) // Do We Have A Rendering Context?  {  if (!wglMakeCurrent(NULL,NULL)) // Are We Able To Release The DC And RC Contexts?  {  MessageBox(NULL,"Release Of DC And RC Failed.","SHUTDOWN ERROR",MB\_OK | MB\_ICONINFORMATION);  }  if (!wglDeleteContext(hRC)) // Are We Able To Delete The RC?  {  MessageBox(NULL,"Release Rendering Context Failed.","SHUTDOWN ERROR",MB\_OK | MB\_ICONINFORMATION);  }  hRC=NULL; // Set RC To NULL  }  if (hDC && !ReleaseDC(hWnd,hDC)) // Are We Able To Release The DC  {  MessageBox(NULL,"Release Device Context Failed.","SHUTDOWN ERROR",MB\_OK | MB\_ICONINFORMATION);  hDC=NULL; // Set DC To NULL  }  if (hWnd && !DestroyWindow(hWnd)) // Are We Able To Destroy The Window?  {  MessageBox(NULL,"Could Not Release hWnd.","SHUTDOWN ERROR",MB\_OK | MB\_ICONINFORMATION);  hWnd=NULL; // Set hWnd To NULL  }  if (!UnregisterClass("OpenGL",hInstance)) // Are We Able To Unregister Class  {  MessageBox(NULL,"Could Not Unregister Class.","SHUTDOWN ERROR",MB\_OK | MB\_ICONINFORMATION);  hInstance=NULL; // Set hInstance To NULL  }  }  /////////////////Mengaktifkan Tampilan Window Fullscreen//////////////////////////////////////  BOOL CreateGLWindow(char\* title, int width, int height, int bits, bool fullscreenflag)  {  GLuint PixelFormat; // Holds The Results After Searching For A Match  WNDCLASS wc; // Windows Class Structure  DWORD dwExStyle; // Window Extended Style  DWORD dwStyle; // Window Style  RECT WindowRect; // Grabs Rectangle Upper Left / Lower Right Values  WindowRect.left=(long)0; // Set Left Value To 0  WindowRect.right=(long)width; // Set Right Value To Requested Width  WindowRect.top=(long)0; // Set Top Value To 0  WindowRect.bottom=(long)height; // Set Bottom Value To Requested Height  fullscreen=fullscreenflag; // Set The Global Fullscreen Flag  hInstance= GetModuleHandle(NULL); // Grab An Instance For Our Window  wc.style= CS\_HREDRAW | CS\_VREDRAW | CS\_OWNDC; // Redraw On Size, And Own DC For Window.  wc.lpfnWndProc=(WNDPROC) WndProc; // WndProc Handles Messages  wc.cbClsExtra=0; // No Extra Window Data  wc.cbWndExtra=0; // No Extra Window Data  wc.hInstance=hInstance; // Set The Instance  wc.hIcon=LoadIcon(NULL, IDI\_WINLOGO); // Load The Default Icon  wc.hCursor=LoadCursor(NULL, IDC\_ARROW); // Load The Arrow Pointer  wc.hbrBackground=NULL; // No Background Required For GL  wc.lpszMenuName=NULL; // We Don't Want A Menu  wc.lpszClassName="OpenGL"; // Set The Class Name  if (!RegisterClass(&wc)) // Attempt To Register The Window Class  {  MessageBox(NULL,"Failed To Register The Window Class.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);  return FALSE; // Return FALSE  }    if (fullscreen) // Attempt Fullscreen Mode?  {  DEVMODE dmScreenSettings; // Device Mode  memset(&dmScreenSettings,0,sizeof(dmScreenSettings)); // Makes Sure Memory's Cleared  dmScreenSettings.dmSize=sizeof(dmScreenSettings); // Size Of The Devmode Structure  dmScreenSettings.dmPelsWidth=width; // Selected Screen Width  dmScreenSettings.dmPelsHeight=height; // Selected Screen Height  dmScreenSettings.dmBitsPerPel=bits; // Selected Bits Per Pixel  dmScreenSettings.dmFields=DM\_BITSPERPEL|DM\_PELSWIDTH|DM\_PELSHEIGHT;  // Try To Set Selected Mode And Get Results. NOTE: CDS\_FULLSCREEN Gets Rid Of Start Bar.  if(ChangeDisplaySettings(&dmScreenSettings,CDS\_FULLSCREEN)!=DISP\_CHANGE\_SUCCESSFUL)  {  // If The Mode Fails, Offer Two Options. Quit Or Use Windowed Mode.  if (MessageBox(NULL,"The Requested Fullscreen Mode Is Not Supported By\nYour Video Card. Use Windowed Mode Instead?","NeHe GL",MB\_YESNO|MB\_ICONEXCLAMATION)==IDYES)  {  fullscreen=FALSE; // Windowed Mode Selected. Fullscreen = FALSE  }  else  {  // Pop Up A Message Box Letting User Know The Program Is Closing.  MessageBox(NULL,"Program Will Now Close.","ERROR",MB\_OK|MB\_ICONSTOP);  return FALSE; // Return FALSE  }  }  }  if (fullscreen) // Are We Still In Fullscreen Mode?  {  dwExStyle=WS\_EX\_APPWINDOW; // Window Extended Style  dwStyle=WS\_POPUP; // Windows Style  ShowCursor(FALSE); // Hide Mouse Pointer  }  else  {  dwExStyle=WS\_EX\_APPWINDOW | WS\_EX\_WINDOWEDGE; // Window Extended Style  dwStyle=WS\_OVERLAPPEDWINDOW; // Windows Style  }  AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle); // Adjust Window To True Requested Size  // Create The Window  if (!(hWnd=CreateWindowEx( dwExStyle, // Extended Style For The Window  "OpenGL", // Class Name  title, // Window Title  dwStyle | // Defined Window Style  WS\_CLIPSIBLINGS | // Required Window Style  WS\_CLIPCHILDREN, // Required Window Style  0, 0, // Window Position  WindowRect.right-WindowRect.left, // Calculate Window Width  WindowRect.bottom-WindowRect.top, // Calculate Window Height  NULL, // No Parent Window  NULL, // No Menu  hInstance, // Instance  NULL))) // Dont Pass Anything To WM\_CREATE  {  KillGLWindow(); // Reset The Display  MessageBox(NULL,"Window Creation Error.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);  return FALSE; // Return FALSE  }  static PIXELFORMATDESCRIPTOR pfd= // pfd Tells Windows How We Want Things To Be  {  sizeof(PIXELFORMATDESCRIPTOR), // Size Of This Pixel Format Descriptor  1, // Version Number  PFD\_DRAW\_TO\_WINDOW | // Format Must Support Window  PFD\_SUPPORT\_OPENGL | // Format Must Support OpenGL  PFD\_DOUBLEBUFFER, // Must Support Double Buffering  PFD\_TYPE\_RGBA, // Request An RGBA Format  bits, // Select Our Color Depth  0, 0, 0, 0, 0, 0, // Color Bits Ignored  0, // No Alpha Buffer  0, // Shift Bit Ignored  0, // No Accumulation Buffer  0, 0, 0, 0, // Accumulation Bits Ignored  16, // 16Bit Z-Buffer (Depth Buffer)  0, // No Stencil Buffer  0, // No Auxiliary Buffer  PFD\_MAIN\_PLANE, // Main Drawing Layer  0, // Reserved  0, 0, 0 // Layer Masks Ignored  };  if (!(hDC=GetDC(hWnd))) // Did We Get A Device Context?  {  KillGLWindow(); // Reset The Display  MessageBox(NULL,"Can't Create A GL Device Context.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);  return FALSE; // Return FALSE  }  if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd))) // Did Windows Find A Matching Pixel Format?  {  KillGLWindow(); // Reset The Display  MessageBox(NULL,"Can't Find A Suitable PixelFormat.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);  return FALSE; // Return FALSE  }  if(!SetPixelFormat(hDC,PixelFormat,&pfd)) // Are We Able To Set The Pixel Format?  {  KillGLWindow(); // Reset The Display  MessageBox(NULL,"Can't Set The PixelFormat.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);  return FALSE; // Return FALSE  }  if (!(hRC=wglCreateContext(hDC))) // Are We Able To Get A Rendering Context?  {  KillGLWindow(); // Reset The Display  MessageBox(NULL,"Can't Create A GL Rendering Context.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);  return FALSE; // Return FALSE  }  if(!wglMakeCurrent(hDC,hRC)) // Try To Activate The Rendering Context  {  KillGLWindow(); // Reset The Display  MessageBox(NULL,"Can't Activate The GL Rendering Context.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);  return FALSE; // Return FALSE  }  ShowWindow(hWnd,SW\_SHOW); // Show The Window  SetForegroundWindow(hWnd); // Slightly Higher Priority  SetFocus(hWnd); // Sets Keyboard Focus To The Window  ReSizeGLScene(width, height); // Set Up Our Perspective GL Screen  if (!InitGL()) // Initialize Our Newly Created GL Window  {  KillGLWindow(); // Reset The Display  MessageBox(NULL,"Initialization Failed.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);  return FALSE; // Return FALSE  }  return TRUE; // Success  }  LRESULT CALLBACK WndProc( HWND hWnd, // Handle For This Window  UINTuMsg, // Message For This Window  WPARAMwParam, // Additional Message Information  LPARAM lParam) // Additional Message Information  {  switch (uMsg) // Check For Windows Messages  {  case WM\_ACTIVATE: // Watch For Window Activate Message  {  if (!HIWORD(wParam)) // Check Minimization State  {  active=TRUE; // Program Is Active  }  else  {  active=FALSE; // Program Is No Longer Active  }  return 0; // Return To The Message Loop  }  case WM\_SYSCOMMAND:  {  switch (wParam)  {  case SC\_SCREENSAVE:  case SC\_MONITORPOWER:  return 0;  }  break;  }  case WM\_CLOSE:  {  PostQuitMessage(0);  return 0;  }  case WM\_KEYDOWN:  {  keys[wParam] = TRUE;  return 0;  }  case WM\_KEYUP:  {  keys[wParam] = FALSE;  return 0;  }  case WM\_SIZE:  {  ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));  return 0;  }  }  return DefWindowProc(hWnd,uMsg,wParam,lParam);  }  //////////////////////////////Prosedur Pertanyaan////////////////////////////////////////////////////////////  int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {  MSG msg;  BOOL done=FALSE;  if (MessageBox(NULL,"Apakah akan dijalankan pada mode layar penuh?", "Mode Layar Penuh?",MB\_YESNO|MB\_ICONQUESTION)==IDNO)  {  fullscreen=FALSE;  }  if (!CreateGLWindow("Kelompok 6 Grafkom IF13",640,480,16,fullscreen))  {  return 0;  }  while(!done)  {  if (PeekMessage(&msg,NULL,0,0,PM\_REMOVE))  {  if (msg.message==WM\_QUIT)  {  done=TRUE;  }  else  {  TranslateMessage(&msg);  DispatchMessage(&msg);  }  }  else  {    if (active)  {  if (keys[VK\_ESCAPE]) // Tekan ESC untuk keluar program  {  done=TRUE;  }  else  {  DrawGLScene(); // Memanggil Prosedur Layar  SwapBuffers(hDC); //  ProcessKeyboard(); // Memanggil Proses Keyboard  // Go back and study again.  }  }  if (keys[VK\_F1]) // Tekan F1 untuk merubah tampilan fullscreen menjadi kecil  {  keys[VK\_F1]=FALSE;  KillGLWindow();  fullscreen=!fullscreen; // Recreate Our OpenGL Window  if (!CreateGLWindow("Kelompok 6 Grafkom IF13",640,480,16,fullscreen))  {  return 0;  }  }  }  }  // Shutdown  KillGLWindow(); // Membunuh Tampiilan  return (msg.wParam); // Keluar dari Program  } |