**MEMBUAT MODEL 3D SISTEM TATA SURYA**

Diajukan untuk Memenuhi Tugas Besar Mata Kuliah Komputer Grafika

**ADANG PURNAMA 10108614**

**MARISA SABATINI SITUMORANG 10108632**

**AGUS WINARTO PRATAMA 10108640**

**CAHYA KHOMARUDIN 10108642**



**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK DAN ILMU KOMPUTER**

**UNIVERSITAS KOMPUTER INDONESIA**

**BANDUNG**

**2012**

**KATA PENGANTAR**

Puji dan syukur kami panjatkan kehadirat Allah Yang Maha Kuasa, karena dengan kehendak-Nya penulis dapat menyelesaikan laporan tugas besar Grafika Komputer ini dengan judul “Membuat Model 3D Sistem Tata Surya” sebagai salah satu syarat untuk memenuhi mata kuliah Grafika Komputer di Universitas Komputer Indonesia.

Penulis menyadari bahwa dalam penulisan laporan ini masih banyak terdapat kekurangan yang harus disempurnakan, sehingga kritik dan saran sangat dibutuhkan demi perbaikan di masa yang akan datang.

Bandung, Juni 2012

Penulis

**BAB I**

**PENDAHULUAN**

* 1. **Latar Belakang Masalah**

Komputer grafika adalah bagian dari ilmu komputer yang berkaitan dengan pembuatan dan manipulasi gambar (visual) secara digital. Bentuk sederhana dari komputer grafika adalah komputer grafika 2D yang kemudian berkembang menjadi komputer 3D, pemrosesan citra (image processing), dan pengenalan pola (pattern recognition). Grafika komputer juga sering dikenal juga dengan istilah visualisasi data.

Seiring berkembangnya grafika komputer, banyak teknologi baru untuk membuat objek 3D, diantaranya adalah OpenGL. OpenGL adalah suatu graphic library yang sebagian bersifat open source, dipakai pada banyak platform (windows, linux) dan dapat digunakan pada berbagai jenis compiler seperti C++ atau Delphi. OpenGL bukanlah bahasa pemrograman tetapi merupakan suatu Application Programming Interface (API).

Berdasarkan penjelasan di atas, maka kami bermaksud untuk membuat model 3D untuk menerapkan komputer grafika dengan teknologi OpenGL, sehinga dapat menambah pemahaman kami tentang komputer grafika dan OpenGL. Objek 3D yang akan kami buat adalah Sistem Tata Surya, dimana ada matahari, planet-planet, dan bintang-bintang. Pada saat planet mengelilingi matahari, tiap planet akan bertranslasi terhadap sumbu y. Matahari akan dibuat berbentuk sphere dan diberi efek lighting sehingga nampak seperti asli, planet-planet dibuat sphere dan berputar-putar, bintang-bintang dibuat dari penggabungan icosahedron, serta dibuat lintasan-lintasan orbit.

* 1. **Perumusan Masalah**

Berdasarkan pemaparan pada latar belakang masalah, maka dapat disimpulkan beberapa masalah :

1. Bagaimana memahami dan menerapkan konsep komputer grafika dalam aplikasi 3D.
2. Bagaimana memahami dan menerapkan fungsi-fungsi OpenGL dalam aplikasi 3D.
   1. **Maksud dan Tujuan**

Untuk mengimplementasikan konsep komputer grafika dan fungsi OpenGL, maka kami bermaksud membuat model 3D Sistem Tata Surya.

Tujuan dari aplikasi ini adalah :

1. Memahami konsep komputer grafika dan fungsi OpenGL dalam model 3D.
2. Menerapkan konsep komputer grafika dan fungsi OpenGL sehingga lebih mengerti.
   1. **Batasan Masalah**

Adapun batasan masalahnya sebagai berikut:

1. Bahasa yang digunakan bahasa pemrograman C++
2. IDE yang digunakan Eclipse C++ dengan library openGL
3. Informasi yang disampaikan hanya mengenai matahari, planet-planet, bintang, dan lintasan orbit yang ada di sistem tata surya.

**BAB II**

**LANDASAN TEORI**

## 2.1 Visual C++

Visual C++ adalah sebuah produk IDE untuk bahasa pemrograman C dan C++ yang dikembangkan Microsoft. 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
  1. **OpenGL**

OpenGL adalah suatu graphic library yang sebagian bersifat open source, dipakai pada banyak platform (windows, linux) dan dapat digunakan pada berbagai jenis compiler seperti C++ atau Delphi. OpenGL bukanlah bahasa pemrograman tetapi merupakan suatu Application Programming Interface (API).

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 | - |

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.

## *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:

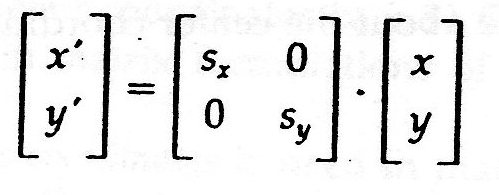
**R = cos(0) sin(0)**

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

## *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 :

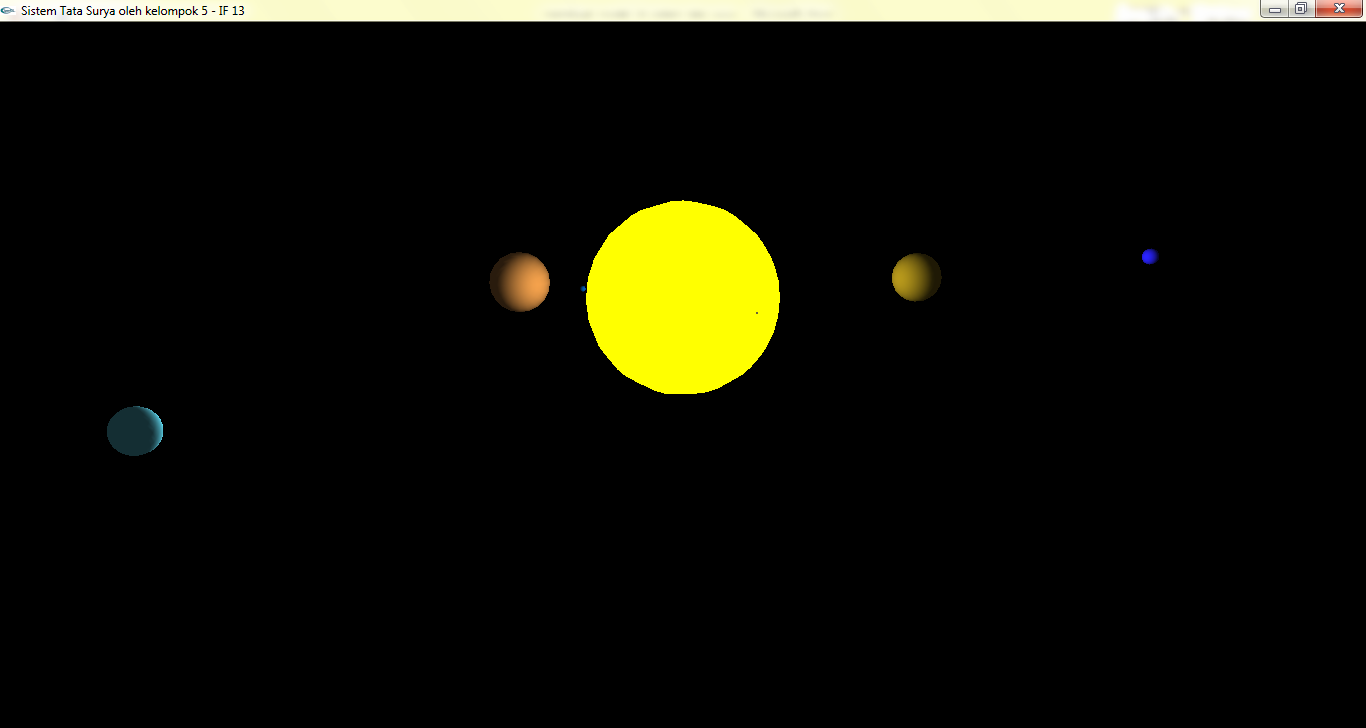


**BAB III**

**IMPLEMENTASI**

* 1. **Tampilan Awal**

Berikut ini merupakan tampilan awal model 3D menggunakan objek Tata Surya dengan menggunakan Visual C++. Tampilan awal dari aplikasi dapat dilihat pada gambar berikut ini.



* 1. **Source Code**

**Tata Surya.cpp**

#include <stdio.h>

#include <stdlib.h>

#include <gl/glut.h>

#include <math.h>

#include "kamera.c"

#include "menu.c"

#include "cahaya.c"

#include "bola.c"

//#include "tekstur.c"

#define TimerMSecs 33

GLenum polygonMode = GL\_FILL;

int SpinOn = 0, SpinBreak = 1, startTime, prevTime, Velocity = 1;

static double Mer = 0.0, Ven = 0.0, Ear = 0.0, Moo = 0.0, Moo2 = 0.0, Mar = 0.0, Jup =

0.0, Sat = 0.0, Ura = 0.0, Nep = 0.0, Plu = 0.0;

GLfloat Shine[] = {10};

GLfloat MooCol[] = {2.3, 2.3, 2.3};

GLfloat Moo2Col[] = {0.93, 0.93, 0.93};

GLfloat MerCol[] = {2.05, 1.90, 1.12};

GLfloat VenCol[] = {1.39, 0.54, 0.26};

GLfloat EarCol[] = {0.0, 0.3, 0.6};

GLfloat MarCol[] = {1.39, 0.26, 0.0};

GLfloat JupCol[] = {0.80, 0.53, 0.25};

GLfloat SatCol[] = {0.6, 0.5, 0.09};

GLfloat UraCol[] ={0.40, 0.900, 1.0};

GLfloat NepCol[] = {0.135, 0.115, 0.85};

GLfloat PluCol[] = {0.7, 0.8, 1.00};

void SceneSpin(int value)

{

int currTime = glutGet(GLUT\_ELAPSED\_TIME);

if(SpinOn == 1)

glutTimerFunc(TimerMSecs, SceneSpin, 0);

Mer += 11.0 \* Velocity;

if(Mer > 360.0)

Mer = Mer - 360.0;

Ven += 7.0 \* Velocity;

if(Ven > 360.0)

Ven = Ven - 360.0;

Ear += 5.0 \* Velocity;

if(Ear > 360.0)

Ear = Ear - 360.0;

Moo += 0.2 \* Velocity;

if(Moo > 360.0)

Moo = Moo - 360.0;

Moo2 += 0.1 \* Velocity;

if(Moo2 > 360.0)

Moo2 = Moo2 - 360.0;

Mar += 3.0 \* Velocity;

if(Mar > 360.0)

Mar = Mar - 360.0;

Jup += 5.0 \* Velocity;

if(Jup > 360.0)

Jup = Jup - 360.0;

Sat += 1.0 \* Velocity;

if(Sat > 360.0)

Sat = Sat - 360.0;

Ura += 2.0 \* Velocity;

if(Ura > 360.0)

Ura = Ura - 360.0;

Nep += 1.0 \* Velocity;

if(Nep > 360.0)

Nep = Nep - 360.0;

Plu += 0.5 \* Velocity;

if(Plu > 360.0)

Plu = Plu - 360.0;

glutPostRedisplay();

prevTime = currTime;

}

void draw(void)

{

GLfloat position[] = { 0.0, 0.0, 1.5, 1.0 };

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SHININESS, Shine);

glPushMatrix();

gluLookAt(80.0, 10.0, 10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glPushMatrix();

//Membuat Merkurius

glRotated((GLdouble) Mer, 0.0, 1.0, 0.0);

glTranslated(0.0, 0.0, 4.9);

glRotated((GLdouble) Mer, 0.0, 1.0, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, MerCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, MerCol);

glutSolidSphere(0.049, 20, 20);

glDisable(GL\_DIFFUSE);

glPopMatrix();

glPushMatrix();

//Membuat Venus

glRotated((GLdouble) Ven, 0.0, 1.0, 0.0);

glTranslated(0.1, 0.0, 5.2);

glRotated((GLdouble) Ven - 1, 0.0, 1.0, 0.0); //Counter Rotation

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, VenCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, VenCol);

glutSolidSphere(0.12, 20, 20);

glDisable(GL\_DIFFUSE);

glPopMatrix();

glPushMatrix();

//Membuat Bumi

glRotated((GLdouble) Ear, 0.0, 1.0, 0.0);

glTranslated(0.1, 0.0, 5.6);

glRotated((GLdouble) Ear, 0.0, 1.0, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, EarCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, EarCol);

glutSolidSphere(0.13, 20, 20);

glDisable(GL\_DIFFUSE);

glPushMatrix();

//Membuat Bulan / Satelit Bumi

glRotated((GLdouble) Moo, 0.0, 1.0, 0.0);

glTranslated(0.1, 0.0, 0.1);

glRotated((GLdouble) Moo, 0.0, 1.0, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, MooCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, MooCol);

glutSolidSphere(0.02, 20, 20);

glDisable(GL\_DIFFUSE);

glPopMatrix();

glPopMatrix();

glPushMatrix();

//Membuat Mars

glRotated((GLdouble) Mar, 0.0, 1.0, 0.0);

glTranslated(0.1, 0.0, 6.0);

glRotated((GLdouble) Mar, 0.0, 1.0, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, MarCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, MarCol);

glutSolidSphere(0.067, 20, 20);

glDisable(GL\_DIFFUSE);

glPushMatrix();

//Membuat Satelit 1 Mars

glRotated((GLdouble) Moo, 0.0, 1.0, 0.0);

glTranslated(0.0, 0.0, 0.1);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, MooCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, MooCol);

glRotated((GLdouble) Moo, 0.0, 1.0, 0.0);

glutSolidSphere(0.015, 20, 10);

glPopMatrix();

glPushMatrix();

//Membuat Satelit 2 Mars

glRotated((GLdouble) Moo2, 0.0, 1.0, 0.0);

glTranslated(0.0, 0.0, 0.15);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, Moo2Col);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, Moo2Col);

glRotated((GLdouble) Moo2, 0.0, 1.0, 0.0);

glutSolidSphere(0.01, 20, 10);

glPopMatrix();

glPopMatrix();

glPushMatrix();

//Membuat Jupiter

glRotated((GLdouble) Jup, 0.0, 1.0, 0.0);

glTranslated(0.1, 0.0, 9.7);

glRotated((GLdouble) Jup, 0.0, 1.0, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, JupCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, JupCol);

glutSolidSphere(1.42, 20, 20);

glDisable(GL\_DIFFUSE);

glPopMatrix();

glPushMatrix();

//Membuat Saturnus

glRotated((GLdouble) Sat, 0.0, 1.0, 0.0);

glTranslated(0.1, 0.0, 14.0);

glRotated((GLdouble) Sat, 0.0, 1.0, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, SatCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, SatCol);

glutSolidSphere(1.20, 20, 20);

glDisable(GL\_DIFFUSE);

glPopMatrix();

glPushMatrix();

//Membuat Uranus

glRotated((GLdouble) Ura, 0.0, 1.0, 0.0);

glTranslated(0.1, 0.0, 23.5);

glRotated((GLdouble) Ura - 1, 1.0, 0.1, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, UraCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, UraCol);

glutSolidSphere(0.51, 20, 20);

glDisable(GL\_DIFFUSE);

glPopMatrix();

glPushMatrix();

//Membuat Neptunus

glRotated((GLdouble) Nep, 0.0, 1.0, 0.0);

glTranslated(0.1, 0.0, 34.5);

glRotated((GLdouble) Nep, 0.0, 1.0, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, NepCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, NepCol);

glutSolidSphere(0.49, 20, 20);

glDisable(GL\_DIFFUSE);

glPopMatrix();

glPushMatrix();

//Membuat Pluto

glRotated((GLdouble) Plu, 0.0, 1.0, 0.0);

glTranslated(0.1, 0.0, 44.0);

glRotated((GLdouble) Plu, 0.0, 1.0, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, PluCol);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, PluCol);

glutSolidSphere(0.1, 20, 20);

glDisable(GL\_DIFFUSE);

glPopMatrix();

glLightfv(GL\_LIGHT0, GL\_POSITION, position);

GLfloat global\_ambient[] = {0.5f, 0.5f, 0.5f, 1.0f}; //Emission

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, global\_ambient); //Emission

glDisable(GL\_LIGHTING);

glColor3f(1.0, 1.0, 0.0);

glRotated((GLdouble) Ear, 0.0, 1.0, 0.0);

glutSolidSphere(4.00, 20, 20);

glEnable(GL\_LIGHTING);

glPopMatrix();

}

void display(void)

{

float angle=0.0;

glClearColor (0.0,0.0,0.0,1.0);

glClear (GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

camera();

enable();

draw();

glutSwapBuffers();

glPolygonMode(GL\_FRONT\_AND\_BACK, polygonMode);

angle++;

}

void WireFrameMode() {

if (polygonMode == GL\_FILL ) {

polygonMode = GL\_LINE;

}

else {

polygonMode = GL\_FILL;

}

}

void Keyboard(unsigned char key, int A, int B)

{

float xpos = 0, ypos = 0, zpos = 0, xrot = 0, yrot = 0;

if(key == 'x')

{

xrot += 1;

if(xrot > 360) xrot -= 360;

}

if(key == 'z')

{

xrot -= 1;

if(xrot <-360) xrot += 360;

}

if(key == 'w')

{

float xrotrad,yrotrad;

yrotrad = (yrot/180 \* 3.141592654f);

xrotrad = (xrot/180 \* 3.141592654f);

xpos += float(sin(yrotrad));

zpos -= float(cos(yrotrad));

ypos -= float(sin(xrotrad));

}

if(key == 's')

{

float xrotrad,yrotrad;

yrotrad = (yrot/180 \* 3.141592654f);

xrotrad = (xrot/180 \* 3.141592654f);

xpos -= float(sin(yrotrad));

zpos += float(cos(yrotrad));

ypos += float(sin(xrotrad));

}

if(key == 'd')

{

yrot+=1;

if(yrot>360) yrot -= 360;

}

if(key == 'a')

{

yrot -= 1;

if(yrot<-360) yrot += 360;

}

switch(key)

{

case '1':SpinOn = 1;

if(SpinBreak == 0){

SpinOn = 0;

}

if(SpinBreak == 1){

glutTimerFunc(TimerMSecs, SceneSpin, 0);

}

if (SpinBreak == 0){

SpinBreak = 1;

}

else SpinBreak = 0;

break;

case '2':SpinOn = 1;

Velocity = Velocity + 1;

break;

case '3':SpinOn = 1;

Velocity = Velocity - 1;

break;

case 27: exit(0);

break;

default:

break;

}

}

void mouseMenu(int value)

{

if(value == 1)

SpinOn = 1;

if(SpinBreak == 0){

SpinOn = 0;

}

if(SpinBreak == 1){

glutTimerFunc(TimerMSecs, SceneSpin, 0);

}

if (SpinBreak == 0){

SpinBreak = 1;

}

else SpinBreak = 0;

if(value == 2)

SceneSpin(2);

if(value == 3)

WireFrameMode();

if(value == 4)

exit(0);

}

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

{

menu();

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowPosition(300, 300);

glutInitWindowSize(500, 500);

glutCreateWindow("Sistem Tata Surya oleh kelompok 5 - IF 13");

glutCreateMenu(mouseMenu);

glutAddMenuEntry("Start/Pause", 1);

glutAddMenuEntry("Langkah Animasi", 2);

glutAddMenuEntry("Bingkai Kawat", 3);

glutAddMenuEntry("Keluar", 4);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glEnable(GL\_LIGHT\_MODEL\_AMBIENT);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(Keyboard);

startTime = glutGet(GLUT\_ELAPSED\_TIME);

prevTime = startTime;

glutMainLoop();

return 0;

}

**cahaya.cpp**

#include <stdio.h>

#include <GL/glut.h>

void enable(void)

{

glShadeModel(GL\_SMOOTH);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_DEPTH\_TEST);

}

**bola.c**

#include <stdio.h>

#include <GL/glut.h>

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei) w, (GLsizei) h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(40.0, (GLfloat) w/(GLfloat) h, 1.0, 500.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

**menu.c**

#include <stdio.h>

void menu(void)

{

printf("--------MENU--------\n");

printf("Tekan 1 untuk start/pause\n");

printf("Tekan 2 untuk meninggkatkan kecepatan\n");

printf("Tekan 3 untuk menurunkan kecepatan\n");

printf("Klik kanan untuk informasi lebih\n\n");

printf("-----KONTROL KAMERA-----\n");

printf("W A S D untuk maju mundur dan kanan kiri\n");

printf("x dan z untuk zoom in dan zoom out\n\n");

printf("-----------------------\n");

printf("Tekan Esc untuk keluar\n");

printf("-----------------------\n");

}

**kamera.c**

#include <GL/glut.h>

void camera(void)

{

float xpos = 0, ypos = 0, zpos = 0, xrot = 0, yrot = 0;

glRotatef(xrot,1.0,0.0,0.0);

glRotatef(yrot,0.0,1.0,0.0);

glTranslated(-xpos,ypos,-zpos);

glutPostRedisplay();

}

**tekstur.c**

#include <windows.h>

#include <gl\gl.h>

#include <gl\glu.h>

#include <glaux.h>

#include <stdio.h>

#include <math.h>

///////////////////////////////////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()

{

GLint loop;

GLuint texture[14];

int Status = FALSE;

AUX\_RGBImageRec \*TextureImage[14];

memset(TextureImage, 0, sizeof(void \*)\*14);

if ( (TextureImage[0]=LoadBMP("earth.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;

}

//////////////////////////////////////////////////////////////////////////////////////////////

**BAB IV**

**KESIMPULAN**

Setelah melakukan perancangan objek 3D yaitu model 3D Sistem Tata Surya, penulis mengambil kesimpulan sebagai berikut :

1. Penulis sedikit mengetahui fungsi-fungsi OpenGL dalam aplikasi 3D.
2. Penulis sedikit mengetahui teknik-teknik 3D dengan beberapa fungsi memberi kesan nyata pada objek yang dibuat.