**APLIKASI OPENGL SUASANA PANTAI (PELABUHAN)**

*Diajukan Untuk Memenuhi Tugas Pengganti UAS Komputer Grafika*

*Dosen Hendri Karisma, S.Kom*



Disusun Oleh:

Benny Murwanto – 10108625

Arief Firmansyah - 10108607

Devi Eka Nainggolan – 10108622

Moch. Dendi Sukandi Yahya - 10107413

Kelas : IF13/VIII

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK DAN ILMU KOMPUTER**

**UNIVERSITAS KOMPUTER INDONESIA**

**BANDUNG**

**2012**

# DAFTAR ISI

[DAFTAR ISI i](#_Toc328547403)

[DAFTAR GAMBAR ii](#_Toc328547404)

[BAB I 1](#_Toc328547405)

[PENDAHULUAN 1](#_Toc328547406)

[1.1. Latar Belakang 1](#_Toc328547407)

[1.2. Maksud dan Tujuan 2](#_Toc328547408)

[1.2.1. Maksud 2](#_Toc328547409)

[1.2.2. Tujuan 2](#_Toc328547410)

[2.1. Batasan Masalah 2](#_Toc328547411)

[BAB II 3](#_Toc328547412)

[LANDASAN TEORI DAN IMPLEMENTASI PROGRAM 3](#_Toc328547413)

[2.1 Pengertian Grafika Komputer 3](#_Toc328547414)

[2.2 Elemen Dasar Grafika 4](#_Toc328547415)

[2.2.1 Point 4](#_Toc328547416)

[2.2.2 Polyline 4](#_Toc328547417)

[2.2.3 Polygon 4](#_Toc328547418)

[2.2.4 Filled Polygon (Face) 4](#_Toc328547419)

[2.2.5 Gradate Polygon 4](#_Toc328547420)

[2.3 Grafik Komputer 2D 4](#_Toc328547421)

[2.4 grafik komputer 3D 5](#_Toc328547422)

[2.5 Perbedaan Grafik 2D dan 3D 5](#_Toc328547423)

[2.6 Transformasi 6](#_Toc328547424)

[2.7 Color (Warna) 9](#_Toc328547425)

[2.8 Lighting (pencahayaan) 10](#_Toc328547426)

[2.9 Blending (Pencampuran) 11](#_Toc328547427)

[2.10 Texture Mapping 12](#_Toc328547428)

[2.10.1 Menentukan Tekstur Image : 12](#_Toc328547429)

[2.10.2 Mendefinisikan gambar sebagai sebuah tekstur 12](#_Toc328547430)

[2.10.3 Mengubah gambar tekstur : 13](#_Toc328547431)

[2.10.4 Mapping Tekstur : 13](#_Toc328547432)

[2.11 Fog 13](#_Toc328547433)

[2.12 Pengertian OpenGl 14](#_Toc328547434)

[2.13 Visual C++ 14](#_Toc328547435)

[2.14 Implementasi 15](#_Toc328547436)

[2.14.1 Rincian Program 15](#_Toc328547437)

[2.14.2 *Capture* Program 16](#_Toc328547438)

[BAB III 17](#_Toc328547439)

[KESIMPULAN DAN SARAN 17](#_Toc328547440)

[5.1 Kesimpulan 17](#_Toc328547441)

[5.2 Saran 17](#_Toc328547442)

[DAFTAR PUSATAKA 18](#_Toc328547443)

[LAMPIRAN 19](#_Toc328547444)

# DAFTAR GAMBAR

[Gambar 1 *Teapot* hasil rendering 3](#_Toc297267739)

[Gambar 2 Pembagian warna 9](#_Toc297267740)

[Gambar 3 *Capture* Program 16](#_Toc297267741)

# BAB I

# PENDAHULUAN

## Latar Belakang

Pemandangan pantai memang indah untuk dilihat, apalagi jika dilihat di pagi hari menjelang matahari terbit dan sore hari menjelang matahari terbenam. Anda juga bisa menikmati hembusan angin pantai yang membuat anda terlelap dalam suasana, dan pemandangan laut. Selain itu juga biasanya di pantai anda bisa melihat kapal laut/ perahu, mercusuar, air laut, pelampung tanda kedalaman laut, dan lain sebagainya.

Dengan memanfaatkan teknologi animasi 3 dimensi, kami ingin mencoba membuat suatu animasi pemandangan pantai untuk mengaplikasikan teknologi 3 dimensi tersebut. Objek-objek yang akan dibangun sebagai berikut:

1. Perahu/kapal laut yang akan berangkat berlayar,
2. Mercusuar sebagai penanda sinyal di daratan,
3. Air laut,
4. Matahari,
5. Dermaga, dan
6. Ikan-ikan di laut.

Kamera akan bergerak 360 derajat mengelilingi pemandangan pantai, secara bersamaan kapal/ perahu beraangkat berlayar, lampu mercusuar berputar 360 derajat, dan perubahan warna pada matahari sebagai penanda pagi, siang dan sore hari.

Berdasarkan hal itu, maka bukan hal mustahil bila pemandangan pantai juga dapat disimulasikan dalam komputer secara 3 dimensi. Salah satu bahasa pemrograman untuk membuat aplikasi 3 dimensi adalah dengan memanfaatkan opengl.

Opengl dapat digunakan untuk membangun aplikasi suasana pantai dengan tools pengembangan software menggunakan devC++

## Maksud dan Tujuan

### Maksud

Maksud dari penulisan ini adalah untuk Membuat aplikasi suasana pantai (pelabuhan) dengan menggunakan OpenGL yang menampilkan objek-objek seperti laut, matahari, mercusuar, dan kapal laut.

### Tujuan

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

1. Menampilkan simulasi keadaan kapal akan brangkat berlayar.
2. Mengetahui fungsi-fungsi pada openGL dengan menggunakan empat unsur yaitu coloring, lighting, blending, dan mapping.

## Batasan Masalah

Agar pembahasan masalah tidak menyimpang dari pokok bahasan, maka batasan masalah dalam pembuatan perangkat lunak ini adalah sebagai berikut :

1. Hanya menampilkan keadaan pantai(pelabuhan) pada saat kapal akan berlayar*.*
2. Menampilkan pergerakan kapal dan rotasi pada mercusuar.
3. Menampilkan rotasi pada pantai
4. Objek ditampilkan dalam 3 dimensi
5. Menggunakan tools Dev C++.

# BAB II

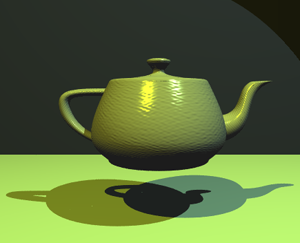
# LANDASAN TEORI DAN IMPLEMENTASI PROGRAM

## 2.1 Pengertian Grafika Komputer

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

Bagian dari grafika komputer meliputi:

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



Gambar 1 *Teapot* hasil rendering

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

## 2.2 Elemen Dasar Grafika

Ada beberapa elemen dasar dari grafika komputer antara lain:

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

### 2.2.1 Point

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

### 2.2.2 Polyline

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

### 2.2.3 Polygon

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

### 2.2.4 Filled Polygon (Face)

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

### 2.2.5 Gradate Polygon

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

## 2.3 Grafik Komputer 2D

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

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

## 2.4 grafik komputer 3D

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

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

## 2.5 Perbedaan Grafik 2D dan 3D

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

## 2.6 Transformasi

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

Jenis jenis dari transformasi dasar:

1. Translasi

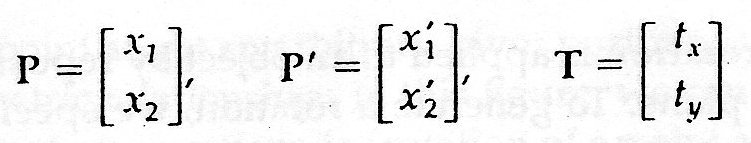
Translasi dilakukan dengan penambahan translasi pada suatu titik koordinat dengan translasi vektor atau shift vektor, yaitu (tx,ty), dimana tx adalah translation vektor menurut sumbu x, sedangkan ty adalah translation vektor menurut sumbu y. koordinat baru titik yang ditranslasi dapat diperoleh dengan

X’= x + tx

Y’= y + ty

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

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



Dengan demikian translasi dua dimensi dapat ditulis dalam bentuk matriks :

|  |
| --- |
| P’ = P + T |

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

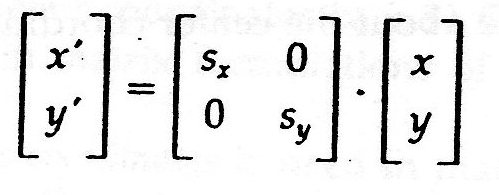
1. Skala

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

X’= x + sx

Y’ = y + sy

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



Dengan demikian skala dapat juga dituliskan

P’ = S . P

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

1. Rotasi

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

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

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

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

sedangkan dengan koordinat polar diketahui bahwa

x = r cos φ, y = r sin φ

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

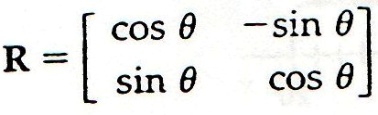
x’ = x cos θ – y sin θ

y’ = x sin θ – y cos θ

matriks transformasi untuk rotasi dapat dinyatakan sebagai berikut:

P’ = R . P

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



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

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

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

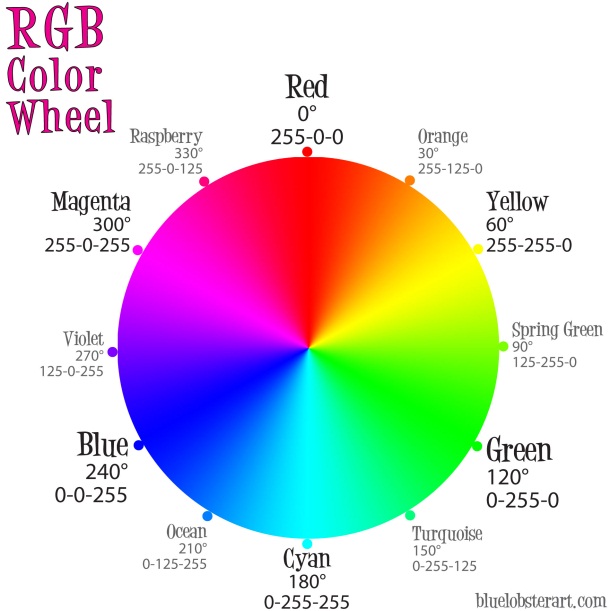
## 2.7 Color (Warna)

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

**Pembagian Warna**

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

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



Gambar 2 Pembagian warna

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

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

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

## 2.8 Lighting (pencahayaan)

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

Beberapa cara mengatasi masalah pencahayaan, antara lain :

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

**` Bayangan**

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

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

## 2.9 Blending (Pencampuran)

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

**Faktor sumber dan tujuan**

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

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

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

## 2.10 Texture Mapping

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

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

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

## 2.10.1 Menentukan Tekstur Image :

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

## 2.10.2 Mendefinisikan gambar sebagai sebuah tekstur

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

Keterangan :

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

## 2.10.3 Mengubah gambar tekstur :

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

## 2.10.4 Mapping Tekstur :

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

## 2.11 Fog

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

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

## 2.12 Pengertian OpenGl

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

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

## 2.13 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

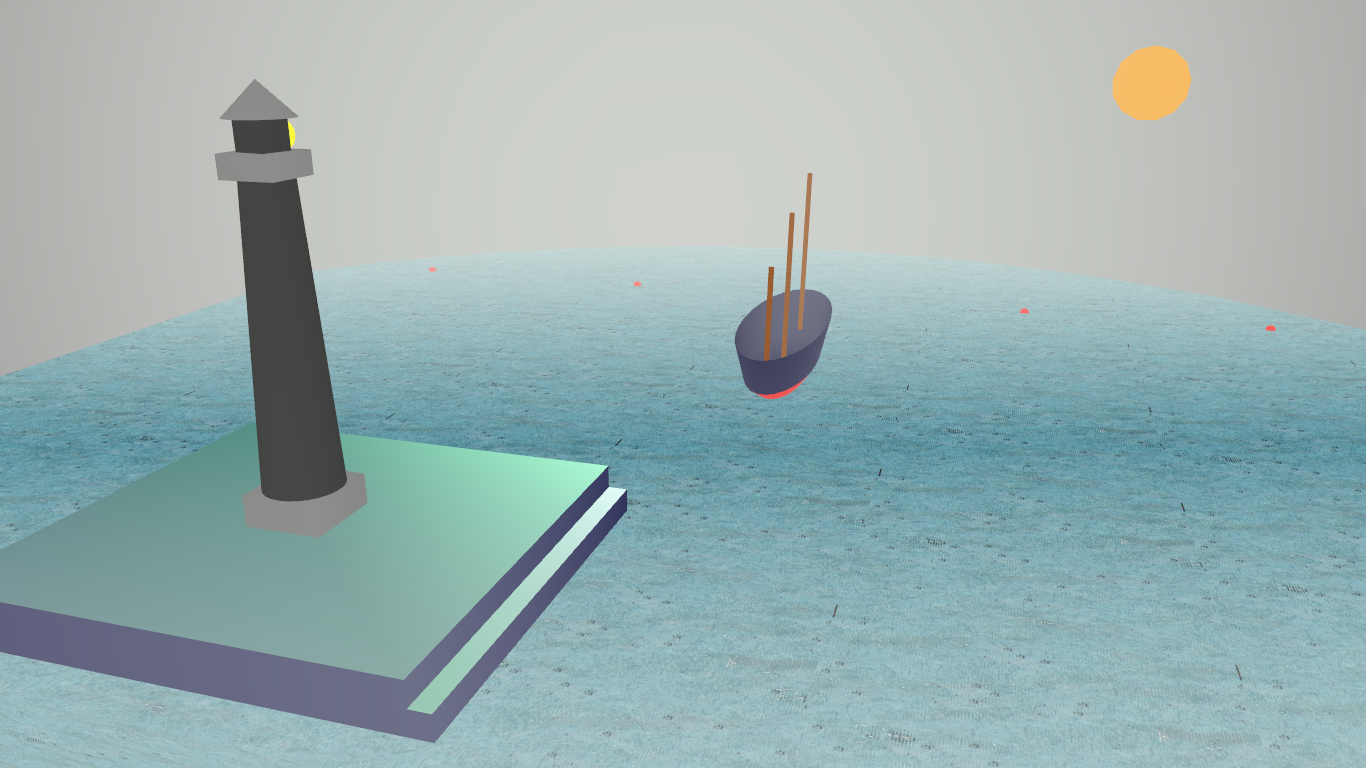
## 2.14 Implementasi

Pembuatan objek yang kami lakukan dilakukan dengan pemanggilan fungsi-fungsi umum yang sudah tersedia di *library* OpenGL itu sendiri. Berikut objek-objek yang kami panggil :

### 2.14.1 Rincian Program

1. Alas yang diberi *texture mepping* laut terbuat dari kotak dengan Vertex, disertai di dalamnya fungsi untuk *looping* *texture mapping.*
2. Langit yang dibuat dari glColor dan campuran dari *texture mapping* berupa langit juga.
3. Dermaga yang dibuat dari dua buah gluSolidCube yaitu fungsi untuk menampilkan kubus, yang ditumpukan dengan tambahan *lighting* atau pencahayaan.
4. Matahari yang dibuat dari gluSolidSphere yaitu fungsi untuk menampilkan lingkaran, dengan pemberian glColor.
5. Perahu yang dibuat dari kombinasi berbagai bangun 2 dimensi.
6. Mercusuar yang dibuat dari ombinasi berbagai bangun 2 dimensi.
7. Efek kabut yang dibuat dengan suatu pemanggilan salah satu fungsi yang ada sehingga menjadi efek kabut.
8. Kombinasi perspektif kamera yang membuat *view* objek semakin nyaman, dan kombinasi transformasi(skala, translasi, dan rotasi), yang membuat objek-objek 2 dimensi yang kami buat seolah-olah seperti 3 dimensi.
9. Ada tambahan *backsound* dengan format file .mp3 yang disimpan di dalam sebuah library dan pencahayaan yang difokuskan hanya pada satu objek saja yaitu dermaga.

### 2.14.2 *Capture* Program



Gambar 3 *Capture* Program

# BAB III

# KESIMPULAN DAN SARAN

## 5.1 Kesimpulan

Kesimpulan merupakan proses terakhir yang berisi hasil dari penelitian yang telah dilakukan. Berikut adalah kesimpulan yang dapat di ambil dari tugas ini :

1. Mengetahui fungsi-fungsi pencampuran lighting, color, blending, fog.
2. Dapat melihat secara langsung perubahan yang terjadi pada objek.

## 5.2 Saran

Kami sangat mengharapkan saran dan kritik dari pembaca yang sifatnya membangun bagi kami untuk memperbaiki segala kekurangan, maka kami dapat menghasilkan makalah yang lebih baik lagi.

# DAFTAR PUSATAKA

http://dhanstar.blogspot.com/2010/10/grafik-komputer-2d-dan-3d.html  
http://aflah7.wordpress.com/2010/10/14/konsep-pemodelan-grafik-2d-dan-3d/  
http://www.ilhamsk.com/desain-pemodelan-grafik/  
http://id.wikipedia.org/wiki/Grafika\_komputer\_3D  
http://id.wikipedia.org/wiki/Grafika\_komputer

# LAMPIRAN

/\*Tugas Besar(UAS) Kelompok 7 IF-13, Komputer Grafika.

Anggota Kelompok:

1. Benny Murwanto - 10108625

2. Arief Firmansyah - 10108607

3. Devi Eka Nainggolan - 10108622

4. Moch. Dendi Sukandi Yahya - 10107413

\*/

#include <cstdlib>

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#include <windows.h>

#include <string.h>

#include <conio2.h>

#include <GL/glut.h>

#include <GL/glext.h>

#include "island.h"

char title[] = "Suasana Pantai 3D",

cmdvol[]="setaudio mySound volume to ",

cmdvolvar[]="";

unsigned int windowWidth= 640,

windowHeight= 480,

windowPosX= 50,

windowPosY= 50,

vol= 300;

bool fullScreenMode= true, kedepan=false;

float sudutputar= 0.0;

unsigned int texture\_id, langit1, langit2, langit3, laut1, laut2, pasir;

static GLfloat lmRot = 0.5f; //rotasi lampu mercusuar

static GLfloat klRot = -5.0; //rotasi kapal laut

//layar kapal laut

GLint nNumPoints = 3;

GLfloat ctrlPoints[3][3][3]= {{{ -4.0f, 0.0f, 4.0f},

{ -2.0f, 4.0f, 4.0f},

{ 4.0f, 0.0f, 4.0f }},

{{ -4.0f, 0.0f, 0.0f},

{ -2.0f, 4.0f, 0.0f},

{ 4.0f, 0.0f, 0.0f }},

{{ -4.0f, 0.0f, -4.0f},

{ -2.0f, 4.0f, -4.0f},

{ 4.0f, 0.0f, -4.0f }}};

// Load texture

int load\_texture(char \*file\_name, unsigned int width, unsigned int height){

GLubyte \*imgbitmap;

FILE \*file;

unsigned short int depth=3;

if ((file = fopen(file\_name, "rb"))==NULL){

printf("File tidak ditemukan: %s!\n",file\_name);

getch(); exit(1);

}

imgbitmap = (GLubyte \*) malloc (width \* height \* depth \* (sizeof(GLubyte)));

if (imgbitmap == NULL){

printf("Alokasi textures ke Memory gagal!\n");

fclose(file);

getch(); exit(1);

}

fread(imgbitmap, width \* height \* depth, 1, file);

fclose(file);

glGenTextures(1, &texture\_id);

glBindTexture(GL\_TEXTURE\_2D, texture\_id);

glPixelStorei(GL\_UNPACK\_ALIGNMENT,1);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);

gluBuild2DMipmaps(GL\_TEXTURE\_2D, GL\_RGB8, width, height, GL\_BGR, GL\_UNSIGNED\_BYTE, imgbitmap);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB8, width, height, 0, GL\_BGR, GL\_UNSIGNED\_BYTE, imgbitmap);

free(imgbitmap);

return texture\_id;

}

// Load all image

void load\_image(){

langit1 = load\_texture("bmp/sky0.bmp", 256, 256);

langit2 = load\_texture("bmp/sky1.bmp", 256, 256);

langit3 = load\_texture("bmp/sky2.bmp", 500, 500);

laut1 = load\_texture("bmp/water2.bmp", 256, 256);

printf("Inisialisasi textures berhasil.\n");

}

// Load background theme song

void load\_sound(){

FILE \*file;

if((file = fopen("bg2.mp3", "r"))==NULL){

printf("File tidak ditemukan: bg.mp3!\n");

getch(); exit(1);

}

else printf("Inisialisasi audio berhasil.\n");

fclose(file);

strcat(cmdvol, itoa(vol, cmdvolvar, 40));

mciSendString("open bg2.mp3 type mpegvideo alias mySound", NULL, 0, 0);

mciSendString("play mySound repeat", NULL, 0, 0);

mciSendString(cmdvol, NULL, 0, 0);

}

// Initialize OpenGL

void init() {

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glClearDepth(100.0f);

glEnable(GL\_DEPTH\_TEST);

glDepthFunc(GL\_LEQUAL);

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

glShadeModel(GL\_SMOOTH);

glEnable(GL\_ALPHA\_TEST);

glAlphaFunc(GL\_GREATER, 0);

load\_image();

load\_sound();

printf("\nJalankan aplikasi.\n");

}

// Draw landscape - laut

void draw\_laut(){

float xleft=-13, y=-1.0, zfar=8.0, xright=11, znear=-20.0, xl, zf, incre=4.0;

zf=zfar;

while (zfar>=znear){

xl=xleft;

while (xl<=xright){

//------------------------ Draw alas

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, laut1);

if(xl==xleft){

glPushMatrix();

glBegin(GL\_QUADS); // Left

glTexCoord2f(1.0f, 1.0f); glVertex3f(xl, y, zfar); // Top Right

glTexCoord2f(1.0f, 0.0f); glVertex3f(xl, y-(incre/4), zfar); // Top Left

glTexCoord2f(0.0f, 0.0f); glVertex3f(xl, y-(incre/4), zfar-incre); // Bottom Left

glTexCoord2f(0.0f, 1.0f); glVertex3f(xl, y, zfar-incre); // Bottom Right

glEnd();

glPopMatrix();

}

if(zfar==zf){

glPushMatrix();

glBegin(GL\_QUADS); // Top

glTexCoord2f(1.0f, 0.0f); glVertex3f(xl, y, zfar); // Top Left

glTexCoord2f(1.0f, 1.0f); glVertex3f(xl+incre, y, zfar); // Top Right

glTexCoord2f(0.0f, 1.0f); glVertex3f(xl+incre, y-(incre/4), zfar); // Bottom Right

glTexCoord2f(0.0f, 0.0f); glVertex3f(xl, y-(incre/4), zfar); // Bottom Left

glEnd();

glPopMatrix();

}

if(xl==xright){

glPushMatrix();

glBegin(GL\_QUADS); // Right

glTexCoord2f(1.0f, 1.0f); glVertex3f(xl+incre, y-(incre/4), zfar); // Top Right

glTexCoord2f(1.0f, 0.0f); glVertex3f(xl+incre, y, zfar); // Top Left

glTexCoord2f(0.0f, 0.0f); glVertex3f(xl+incre, y, zfar-incre); // Bottom Left

glTexCoord2f(0.0f, 1.0f); glVertex3f(xl+incre, y-(incre/4), zfar-incre); // Bottom Right

glEnd();

glPopMatrix();

}

if(zfar==znear){

glPushMatrix();

glBegin(GL\_QUADS); // Bottom

glTexCoord2f(1.0f, 0.0f); glVertex3f(xl, y, zfar-incre); // Top Left

glTexCoord2f(1.0f, 1.0f); glVertex3f(xl+incre, y, zfar-incre); // Top Right

glTexCoord2f(0.0f, 1.0f); glVertex3f(xl+incre, y-(incre/4), zfar-incre); // Bottom Right

glTexCoord2f(0.0f, 0.0f); glVertex3f(xl, y-(incre/4), zfar-incre); // Bottom Left

glEnd();

glPopMatrix();

}

glDisable(GL\_TEXTURE\_2D);

//------------------------ Draw laut

glPushMatrix();

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, laut1);

glBegin(GL\_QUADS);

glTexCoord2f(1.0f, 0.0f); glVertex3f(xl, y, zfar); // Top Left

glTexCoord2f(1.0f, 1.0f); glVertex3f(xl+incre, y, zfar); // Top Right

glTexCoord2f(0.0f, 1.0f); glVertex3f(xl+incre, y, zfar-incre); // Bottom Right

glTexCoord2f(0.0f, 0.0f); glVertex3f(xl, y, zfar-incre); // Bottom Left

glEnd();

glDisable(GL\_TEXTURE\_2D);

glPopMatrix();

xl+=incre;

}

zfar-=incre;

}

}

// Draw langit

void draw\_langit(){

glPushMatrix();

glColor3ub(125, 125, 125);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, langit3);

GLUquadricObj\* q = gluNewQuadric();

gluQuadricDrawStyle(q, GLU\_FILL);

gluQuadricNormals(q, GLU\_SMOOTH);

gluQuadricTexture(q, GL\_TRUE);

glRotatef(90.0, 1.0, 0.0, 0.0);

glScalef(16.0, 16.0, 16.0);

gluSphere(q, 1.0, 30, 30);

glDisable(GL\_TEXTURE\_2D);

glPopMatrix();

}

// Draw Pulau

void draw\_pulau(){

glPushMatrix();

GLfloat pos[4]={5.0,18.0,-15.0,70.0};

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

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

//Alas 1

glPushMatrix();

glTranslatef(-3.0, -0.8, 5.0);

glScalef(6.0, 0.3, 5.0);

glColor3ub(101, 49, 10);

glutSolidCube(1.0);

glPopMatrix();

//Alas 2

glPushMatrix();

glTranslatef(-3.0, -0.5, 5.0);

glScalef(5.5, 0.3, 5.0);

glColor3ub(236, 155, 74);

glutSolidCube(1.0);

glPopMatrix();

glDisable(GL\_LIGHT0);

glDisable(GL\_LIGHTING);

glPopMatrix();

}

void draw\_mercusuar(){

GLUquadricObj \*pObj; // Quadric object

pObj =gluNewQuadric();

gluQuadricNormals(pObj, GLU\_SMOOTH);

glPushMatrix();

glTranslatef(-3.0, 3.7, 5.0);

glScalef(0.19, 0.19, 0.19);

//uppest part

glPushMatrix();

glTranslatef(0.0f, 2.50f, 0.0f);

glRotatef(-90, 1.0f, 0.0f, 0.0f);

glColor3ub(100, 100, 100);

gluCylinder(pObj, 2.0f, 0.0f, 2.0f, 26, 1);

gluDisk(pObj, 0.0f, 2.0f, 26, 13);

glPopMatrix();

//lamp

glPushMatrix();

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

glTranslatef(0.0f, 1.50f, 0.0f);

glColor3ub(0, 0, 0);

glutSolidCube(2);

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

// Specify new position and direction in rotated coords.

glColor3ub(255,255,0);

gluSphere(pObj, 1.0f, 26, 13);

glPopMatrix();

//upper cube

glPushMatrix();

glScalef(1.0f, 0.4f, 1.0f);

glColor3ub(100, 100, 100);

glutSolidCube(3.5);

glPopMatrix();

//cylinder part

glPushMatrix();

glTranslatef(0.0f, -20.0f, 0.0f);

glRotatef(-90, 1.0f, 0.0f, 0.0f);

glColor3ub(0, 0, 0);

gluCylinder(pObj, 2.50f, 1.50f, 20.0f, 26, 1);

glPopMatrix();

//lowest part

glPushMatrix();

glTranslatef(0.0f, -20.0f, 0.0f);

glScalef(1.0f, 0.4f, 1.0f);

glColor3ub(100, 100, 100);

glutSolidCube(5);

glPopMatrix();

glPopMatrix();

}

void draw\_kapal\_laut(){

GLUquadricObj \*pObj; // Quadric object

pObj =gluNewQuadric();

gluQuadricNormals(pObj, GLU\_SMOOTH);

M3DVector3f vCorners[11] = { { 0.0f, 0.0f, 0.0f },// 0 //for the layar

{ 3.0f, -5.0f, 0.0f }, // 1

{ -3.0f, -5.0f, 0.0f }, // 2

{20.0f, 20.0f, 0.0f}, // 3 //for the sea

{20.0f, -20.0f, 0.0f}, //4

{-20.0f, 20.0f, 0.0f}, //5

{-20.0f, -20.0f, 0.0f},//6

{10.0f, 10.0f, 2.0f},//7 //for the pic

{10.0f, 10.0f, -2.0f},//8

{10.0f, 14.0f, -2.0f},//9

{10.0f, 14.0f, 2.0f}//10

};

glPushMatrix();

glRotatef(90, 0.4, 12.0, 0.0);

glTranslatef(klRot++/45, -1.2, 0.8);

glScalef(0.15, 0.15, 0.15);

glPushMatrix();//first

glRotatef(-180, 0.0f, 1.0f, 1.0f);

glColor3ub( 238,59,59);

glScalef(3.0, 1.0f, 1.0f );

gluCylinder(pObj, 2.0f, 4.0f, 3.0f, 26, 1);

gluCylinder(pObj, 0.0f, 2.0f, 0.10f, 26, 1);

glColor3ub(47, 47, 79);

glTranslatef(0.0f, 0.0f, 3.0f);

gluCylinder(pObj, 4.0f, 5.0f, 5.0f, 26, 1);

gluDisk(pObj, 0.0f, 4.0f, 26, 13);

glPopMatrix();//first

glPushMatrix();//second

glColor3ub(139, 69, 19);

glRotatef(-180, 0.0f, 1.0f, 1.0f);

glTranslatef(10.0f, 0.0f, 3.0f); //first stick

gluCylinder(pObj, 0.30f, 0.30f, 14.0f, 26, 1);

glTranslatef(0.0f, 0.0f, 14.0f);

gluDisk(pObj, 0.0f, 0.30f, 26, 13);

glTranslatef(-10.0f, 0.0f, -14.0f); //second stick

gluCylinder(pObj, 0.30f, 0.30f, 18.0f, 26, 1);

glTranslatef(0.0f, 0.0f, 18.0f);

gluDisk(pObj, 0.0f, 0.30f, 26, 13);

glTranslatef(-10.0f, 0.0f, -17.0f); //third stick

gluCylinder(pObj, 0.30f, 0.30f, 20.0f, 26, 1);

glTranslatef(0.0f, 0.0f, 20.0f);

gluDisk(pObj, 0.0f, 0.30f, 26, 13);

glPopMatrix();//second

glPushMatrix();//third

glColor3ub(0, 0, 0);

glTranslatef(-0.40f, 15.0f, 0.0f);

glScalef(1.50f, 1.30f, 1.0f);

glRotatef(-180, 1.0f, 1.0f, 0.0f);

// Sets up the bezier

// This actually only needs to be called once and could go in

// the setup function

glMap2f(GL\_MAP2\_VERTEX\_3, // Type of data generated

0.0f, // Lower u range

10.0f, // Upper u range

3, // Distance between points in the data

3, // Dimension in u direction (order)

0.0f, // Lover v range

10.0f, // Upper v range

9, // Distance between points in the data

3, // Dimension in v direction (order)

&ctrlPoints[0][0][0]); // array of control points

// Enable the evaluator

// Map a grid of 10 points from 0 to 10

glMapGrid2f(10,0.0f,10.0f,10,0.0f,10.0f);

// Evaluate the grid, using lines

glEvalMesh2(GL\_FILL,0,10,0,10);

glPopMatrix();//third

glPushMatrix();//fourth

glTranslatef(9.70f, 15.0f, 0.0f);

glScalef(1.50f, 1.5f, 1.0f);

glRotatef(-180, 1.0f, 1.0f, 0.0f);

glMap2f(GL\_MAP2\_VERTEX\_3, 0.0f, 10.0f, 3, 3, 0.0f, 10.0f, 9,3, &ctrlPoints[0][0][0]);

glMapGrid2f(10,0.0f,10.0f,10,0.0f,10.0f);

glEvalMesh2(GL\_FILL,0,10,0,10);

glPopMatrix();//fourth

glPushMatrix();//fifth

glTranslatef(-10.50f, 13.0f, 0.0f);

glScalef(1.30f, 1.0f, 1.0f);

glRotatef(-180, 1.0f, 1.0f, 0.0f);

glMap2f(GL\_MAP2\_VERTEX\_3, 0.0f, 10.0f, 3, 3, 0.0f, 10.0f, 9,3, &ctrlPoints[0][0][0]);

glMapGrid2f(10,0.0f,10.0f,10,0.0f,10.0f);

glEvalMesh2(GL\_FILL,0,10,0,10);

glPopMatrix();//fifth

glPushMatrix();//sixth

glTranslatef(0.30f, 10.0f, 0.0f);

glColor3ub(255, 255, 255);

glPopMatrix();//sixth

glPopMatrix();

if(klRot>650) klRot=-350;

}

void draw\_matahari(){

glPushMatrix();

//glTranslatef(-3.0, -0.8, 5.0);

glTranslatef(7.0, 4.0, -10.0);

glScalef(0.15, 0.15, 0.15);

glColor3ub(255, 147, 0);

glutSolidSphere(5.0f, 13, 26);

glPopMatrix();

}

// Handler for window paint and re-paint event

void display(void) {

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

GLfloat pos0[4] ={5.0, 5.0, 5.0, 0.0};

GLfloat ambient[4] = { 0.0, 0.0, 1.0, 1.0 };

GLfloat diffuse[4] = { 1.0, 2.0, 1.0, 1.0 };

GLfloat specular[4] = { 1.0, 1.0, 0.0, 1.0 };

GLfloat shininess[1] = { 10.0 };

glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuse);

glLightfv(GL\_LIGHT0, GL\_POSITION, pos0);

glEnable(GL\_BLEND);

glBlendFunc(GL\_ONE, GL\_ONE);

GLfloat kabut[4]={0.93,0.95,0.92,0.88};

glFogfv(GL\_FOG\_COLOR, kabut);

//ketebalan kabut

glFogf(GL\_FOG\_DENSITY, 0.08f);

//glFogf(GL\_FOG\_START, 3.0f);

//glFogf(GL\_FOG\_END, -3.0f);

glFogi(GL\_FOG\_MODE, GL\_EXP);

glEnable(GL\_FOG);

glDisable(GL\_BLEND);

glLoadIdentity();

glRotatef(sudutputar, 0.0f, 1.0f, 0.0f); // Rotate over y-axis

draw\_laut();

draw\_langit();

draw\_pulau();

draw\_mercusuar();

draw\_kapal\_laut();

draw\_matahari();

//pelampung

glPushMatrix();

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

glTranslatef (-4.25, -1.0, -10.0);

glScalef (0.1, 0.1, 0.1);

glutSolidSphere(1.0, 16, 16);

glPopMatrix();

glPushMatrix();

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

glTranslatef (-10.0, -1.0, -10.0);

glScalef (0.1, 0.1, 0.1);

glutSolidSphere(1.0, 16, 16);

glPopMatrix();

glPushMatrix();

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

glTranslatef (5.0, -1.0, -10.0);

glScalef (0.1, 0.1, 0.1);

glutSolidSphere(1.0, 16, 16);

glPopMatrix();

glPushMatrix();

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

glTranslatef (10.0, -1.0, -10.0);

glScalef (0.1, 0.1, 0.1);

glutSolidSphere(1.0, 16, 16);

glPopMatrix();

glutSwapBuffers();

sudutputar -= 0.1f;

if(sudutputar<-360) sudutputar=360;

}

// Handler for window's re-size event

void reshape(GLsizei width, GLsizei height) {

if (height == 0) height = 1;

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

//glOrtho(-10,10,-10,10,-20,20);

gluPerspective(40.0, (float)width / (float)height, 0.1, 100.0);

gluLookAt(0.0, 5.0, 15.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

// Handler for key event

void keyboard(unsigned char key, int x, int y) {

switch (key) {

case 27: exit(0); break; // ESC: exit the program

default: break;

}

}

// Handler for special-key event

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

char varvol[]="setaudio mySound volume to ";

switch (key) {

case GLUT\_KEY\_F1: // F1: Toggle between full-screen and windowed mode

fullScreenMode = !fullScreenMode;

if (fullScreenMode) {

windowPosX = glutGet(GLUT\_WINDOW\_X);

windowPosY = glutGet(GLUT\_WINDOW\_Y);

windowWidth = glutGet(GLUT\_WINDOW\_WIDTH);

windowHeight = glutGet(GLUT\_WINDOW\_HEIGHT);

glutFullScreen();

} else {

glutReshapeWindow(windowWidth, windowHeight);

glutPositionWindow(windowPosX, windowPosX);

}

break;

case GLUT\_KEY\_F2:

if (vol>0){

vol-=50;

strcat(varvol, itoa(vol, cmdvolvar, 10));

mciSendString(varvol, NULL, 0, 0);

}

break;

case GLUT\_KEY\_F3:

if (vol<1000){

vol+=50;

strcat(varvol, itoa(vol, cmdvolvar, 10));

mciSendString(varvol, NULL, 0, 0);

}

break;

case GLUT\_KEY\_F4:

mciSendString("pause mySound", NULL, 0, 0);

break;

case GLUT\_KEY\_F5:

mciSendString("resume mySound", NULL, 0, 0);

break;

default: break;

}

}

// main function: GLUT runs as a console application

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

glutInit(&argc, argv);

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

glutInitWindowSize(windowWidth, windowHeight);

glutInitWindowPosition(windowPosX, windowPosY);

glutCreateWindow(title);

glutFullScreen();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

glutSpecialFunc(specialKey);

glutIdleFunc(display);

init();

glutMainLoop();

return 0;

}

Island.h

typedef float M3DVector3f[18];

void m3dCrossProduct(M3DVector3f result, const M3DVector3f u, const M3DVector3f v)

{

result[0] = u[1]\*v[2] - v[1]\*u[2];

result[1] = -u[0]\*v[2] + v[0]\*u[2];

result[2] = u[0]\*v[1] - v[0]\*u[1];

}

void m3dFindNormal(M3DVector3f result, const M3DVector3f point1, const M3DVector3f point2,

const M3DVector3f point3)

{

M3DVector3f v1,v2; // Temporary vectors

// Calculate two vectors from the three points. Assumes counter clockwise

// winding!

v1[0] = point1[0] - point2[0];

v1[1] = point1[1] - point2[1];

v1[2] = point1[2] - point2[2];

v2[0] = point2[0] - point3[0];

v2[1] = point2[1] - point3[1];

v2[2] = point2[2] - point3[2];

// Take the cross product of the two vectors to get

// the normal vector.

m3dCrossProduct(result, v1, v2);

}