**TUGAS BESAR GRAFIKA KOMPUTER**

**KANDANG HAMSTER 3D**

Diajukan untuk memenuhi

Tugas Mata Kuliah Grafika Komputer

Program Strata Satu Jurusan Teknik Informatika

Fakultas Teknik dan Ilmu Komputer

Universitas Komputer Indonesia

Dosen : Hendri Karisma, S. Kom

**KELOMPOK 8**

**OLEH :**

**ASEP ROJALI 10109411**

**ANGGI SOFYAN 10109408**

**EKA WIBAWA 10109407**



**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK DAN ILMU KOMPUTER**

**UNIVERSITAS KOMPUTER INDONESIA**

**BANDUNG**

**2012**

1. **Latar Belakang**

Kandang *hamster* saat ini semakin bervariasi dari mulai bahan, aksesoris dan ukuran, seperti bahan kawat atau logam, bahan kaca (aquarium), plastik dan bahan kayu dengan ukuran secara umum 75 x 50 x 25 cm untuk kandang besar, 38 x 25 x 25 untuk kandang kecil dan 40 x 20 x 12 cm untuk kandang sedang. Di kandang *hamster* tersebut terdapat berbagai macam aksesoris mainan seperti *jogging wheel,* bola , terowongan, ranjang tidur, pot tanaman, tangga, rumah dan sebagainya.

Untuk memenuhi salah satu tugas mata kuliah komputer grafik, kami akan mengimplementasikan kandang hamster beserta aksesorisnya ke dalam bentuk 3D, dimana objek yang akan di tuangkan ke dalam bentuk 3D adalah pot tanaman, tangga, tempat minum , tempat makan, ranjang tidur, jogging wheel, dan serbuk kayu.

Sehingga pada tugas ini kami beri judul “KANDANG HAMSTER 3D ”.

1. **Rumusan Masalah**

Rumusan masalah pada pembangunan kandang *hamster* 3D adalah sebagai berikut :

1. Bagaimana membuat kandang hamster 3D?
2. Bagaimana mengimplementasikan bahan ajar mata kuliah Komputer Grafika dalam membuat bangun ruang dengan menggunakan OpenGL?
3. **Batasan Masalah**

Agar pembahasan masalah tidak menyimpang dari pokok bahasan, maka batasan masalah dalam pembuatan kandang hamster 3D adalah :

1. Hanya menampilkan pohon, tangga, tempat minum, tempat tidur, tempat makan, *jogging wheel ,* rumah.
2. Objek ditampilkan ke dalam 3D.
3. **List Of Object**
4. Pohon `
5. Tangga
6. Tempat minum
7. Tempat tidur
8. Mainan berputar hamster (*jogging wheel*)
9. Rumah
10. Tempat makan
11. **List Object Pembangun :**

a.Pot tanaman yang dibuat dari objek lingkaran solid yang ditinggikan pada kordinat z- nya, tanaman diatas pot dibuat dengan mengabungkan kubus dengan kerucut.

b.Tangga di buat dengan kubus yang ditransformasikan sedimikian rupa sehingga terlihat seperti tangga.

c. Tempat minum dibuat dari gabungan dua lingkaran.

d. tempat tidur dibuat dari gabungan kubus-kubus.

e. Mainan berputar hamster dibuat dengan gabungan lingkaran

f. Rumah dibuat dari gabungan kubus, segitiga.

g. Tempat makan dibuat dari gabungan kubus-kubus.

**BAB II**

## 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.

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

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

1. Polyline

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

1. Polygon

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

1. Filled Polygon ( Face )

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

1. Gradate Polygon

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

## 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 sebagai berikut :

1. Translasi

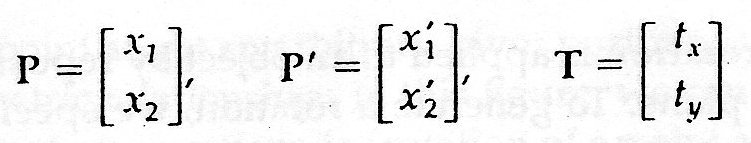
Translasi dilakukan dengan penambahan translasi pada suatu titik koordinat dengan translasi vektor atau shift vektor, yaitu (tx,ty), dimana tx adalah translation vektor menurut sumbu x, sedangkan ty adalah translation vektor menurut sumbu y. koordinat baru titik yang ditranslasi dapat diperoleh dengan

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.

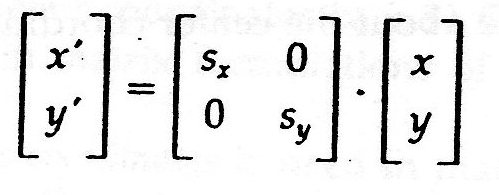
1. Skala

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:

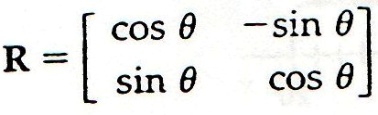
x’ = x cos θ – y sin θ

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

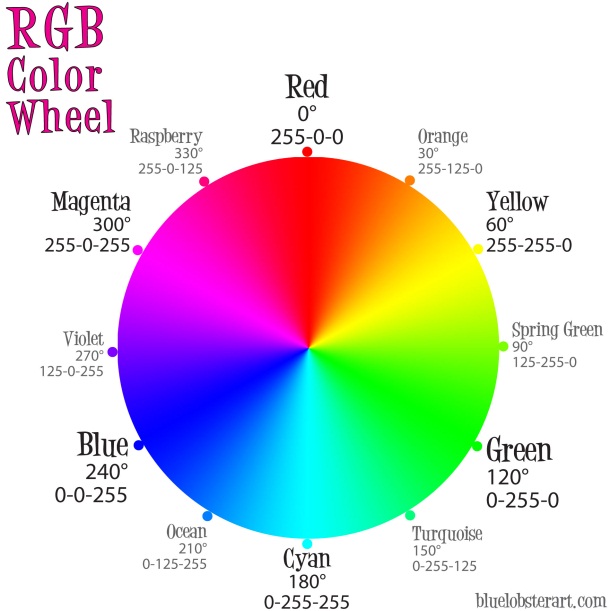
Y’ = yp + (x – xp) sin θ + (y – yp) cos θ

## 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 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 seperti : Wrapping , filtering, dsb.

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

## 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

## 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).

## 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.

## 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.

**BAB III**

* 1. **Source Program**
* main.cpp

/\*

\* main.cpp

\*

\* Created on: Jun 27, 2013

\* Author: kodehijau

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <GL/glut.h>

#include <GL/glu.h>

#include <GL/gl.h>

#include "imageloader.h"

#include "vec3f.h"

static GLfloat spin = 0.0;

float angle = 0;

GLuint texture[2];

float lastx, lasty;

GLint stencilBits;

static int viewx = 50;

static int viewy = 24;

static int viewz = 80;

float rot = 0;

void initRendering() { //inisisalisasi

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

glEnable(GL\_COLOR\_MATERIAL); //warna

glEnable(GL\_LIGHTING); //cahaya

glEnable(GL\_LIGHT0); //lampu

glEnable(GL\_NORMALIZE);

glShadeModel(GL\_SMOOTH);

}

float \_angle = 60.0f;

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

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

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

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

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

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

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

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

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

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

void cylinder(float alas, float atas, float tinggi) {

float i;

glPushMatrix();

glTranslatef(1.0, 0.0, -alas / 8);

glutSolidCone(alas, 0, 32, 4);

for (i = 0; i <= tinggi; i += alas / 24) {

glTranslatef(0.0, 0.0, alas / 24);

glutSolidTorus(alas / 4, alas - ((i \* (alas - atas)) / tinggi), 16, 16);

}

glTranslatef(0.0, 0.0, alas / 4);

glutSolidCone(atas, 0, 20, 1);

glPopMatrix();

}

void patokan(int panjang) {

int x;

glPushMatrix();

for (x = 0; x < panjang; x++) {

glutSolidCube(10);

glTranslated(10.0, 0.0, 0.0);

}

glPopMatrix();

glPushMatrix();

for (x = 0; x < panjang; x++) {

glTranslated(-10.0, 0.0, 0.0);

glutSolidCube(10);

}

glPopMatrix();

}

void bawahKandang(int lebar) {

int zplus;

int zmin;

patokan(10);

glPushMatrix();

for (zplus = 0; zplus < lebar; ++zplus) {

glTranslated(0.0, 0.0, 10.0);

patokan(10);

}

glPopMatrix();

glPushMatrix();

for (zmin = 0; zmin < lebar; ++zmin) {

glTranslated(0.0, 0.0, -10.0);

patokan(10);

}

glPopMatrix();

}

void pohon() {

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.0, 1.0, 0.0);

glRotated(-90.0, 1.0, 0.0, 0.0);

glutSolidCone(15.0, 15.0, 15, 10);

glPopMatrix();

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.803921568627451, 0.5215686274509804, 0.2470588235294118);

glRotated(-90.0, 1.0, 0.0, 0.0);

glTranslated(-1.0, 0.0, -20.0);

cylinder(8.0, 8.0, 20.0);

glPopMatrix();

}

void segitiga() {

glBegin(GL\_QUADS);

glVertex3f(-2, -2, 2);

glVertex3f(-2, -2, -2);

glVertex3f(-2, 2, -2);

glVertex3f(-2, 2, 2);

glEnd();

//Sisi-sisi Prisma

glBegin(GL\_TRIANGLES);

glColor3d(1.0f, 1.0f, 1.0f);

//Segitiga Warna Merah

glVertex3f(-2, -2, 2);

glVertex3f(8, 0, 0);

glVertex3f(-2, 2, 2);

//Segitiga Warna Hijau

glVertex3f(-2, 2, 2);

glVertex3f(8, 0, 0);

glVertex3f(-2, 2, -2);

//Segitiga Warna Biru

glVertex3f(-2, 2, -2);

glVertex3f(8, 0, 0);

glVertex3f(-2, -2, -2);

//Segitiga Warna Putih

glVertex3f(-2, -2, -2);

glVertex3f(8, 0, 0);

glVertex3f(-2, -2, 2);

glEnd();

}

void rumahMewah() {

//atap

glPushMatrix();

glTranslated(0.0, 40.0, 16.0);

glRotated(90.0, 0.0, 0.0, 1.0);

glScaled(2.0, 15.0, 15.0);

segitiga();

glPopMatrix();

//rumah bawah atas

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.903921568627451, 0.5215686274509804, 0.2470588235294118);

glTranslated(0.0, 30.0, 17.5);

glutSolidCube(25);

glPopMatrix();

//rumah bawah

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.803921568627451, 0.5215686274509804, 0.2470588235294118);

glTranslated(0.0, 10.0, 0.0); //cube

//glScaled(1.5, 1.5, 1.5);

glutSolidCube(30);

glTranslated(0.0, 0.0, 30.0);

glutSolidCube(30);

glPopMatrix();

}

void muterMuteran() {

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.903921568627451, 0.5215686274509804, 0.2470588235294118);

glTranslated(00.0, 20.0, -1.0);

glRotated(90, 1.0, 0.0, 0.0);

glRotated(30.0, 1.0, 0.0, 0.0);

cylinder(3.0, 3.0, 20.0);

glPopMatrix();

//kaki

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.903921568627451, 0.5215686274509804, 0.2470588235294118);

glTranslated(00.0, 20.0, 1.0);

glRotated(90, 1.0, 0.0, 0.0);

glRotated(-30.0, 1.0, 0.0, 0.0);

cylinder(3.0, 3.0, 20.0);

glPopMatrix();

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.0, 0.0, 0.0);

glutSolidCube(10);

glTranslated(0.0, 0.0, 10.0);

glutSolidCube(10);

glTranslated(0.0, 0.0, -20.0);

glutSolidCube(10);

glPopMatrix();

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.1, 1.0, 1.0);

glTranslated(0.0, 40.0, 0.0);

glRotated(-90, 0.0, 1.0, 0.0);

cylinder(20.0, 20.0, 5.0);

glPopMatrix();

}

void atapRumah() {

glPushMatrix();

glRotated(90, 0.0, 0.0, 1.0);

glScaled(5.0, 25.0, 25.0);

segitiga();

glPopMatrix();

}

void tempatminum() {

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.803921568627451, 0.5215686274509804, 0.2470588235294118);

glTranslated(0.0, 0.5, 0.0);

glutSolidCube(25);

glTranslated(0.0, 20.0, 0.0);

glutSolidCube(20);

glTranslated(0.0, 10, 0.0);

glutSolidCube(20);

glPopMatrix();

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.1, 1.0, 1.0);

glTranslated(0. - 1, 43.0, 0.0);

glRotated(-90.0, 1.0, 0.0, 0.0);

cylinder(10.0, 10.0, 15.0);

glPopMatrix();

glPushMatrix();

glTranslated(5.0, 27.0, 0.0);

glRotated(90.0, 0.0, 1.0, 0.0);

glRotated(40.0, 1.0, 0.0, 0.0);

cylinder(1.0, 1.0, 15.0);

glPopMatrix();

}

void display(void) {

glClearStencil(0); //clear the stencil buffer

glClearDepth(1.0f);

glClearColor(0.0, 0.6, 0.8, 1);

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

glLoadIdentity();

gluLookAt(viewx, viewy, viewz, 0.0, 0.0, 5.0, 0.0, 1.0, 0.0);

//Tambahin Objek dibawah ini

//Tangga

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.803921568627451, 0.5215686274509804, 0.2470588235294118);

glRotated(90, 0, 1, 0);

//glTranslated(55, 20, 60);

glTranslated(15, 0, 50);

cylinder(3, 3, 10);

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

glTranslated(11, 10, 0);

cylinder(5, 5, 10);

}

glPopMatrix();

//Tempat Minum galon

glPushMatrix();

glTranslated(20.0, 0.0, -85.0);

glRotated(-90.0, 0.0, 1.0, 0.0);

glTranslated(173.0, 0.0, -20.0);

glRotated(90.0, 0.0, 1.0, 0.0);

glRotated(90.0, 0.0, 1.0, 0.0);

tempatminum();

glPopMatrix();

//Rumah

glPushMatrix();

//glRotated(90.0, 1.0, 0.0, 0.0);

glTranslated(40.0, 1.0, -70.0);

glRotated(90.0, 0.0, 1.0, 0.0);

rumahMewah();

glPopMatrix();

//Pagar di tengah

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.803921568627451, 0.5215686274509804, 0.2470588235294118);

glRotated(-90.0, 1.0, 0.0, 0.0);

glTranslated(0.0, -95.0, 5.0);

cylinder(6.0, 6.0, 15.0);

glTranslated(4.0, 15.0, 0.0);

cylinder(6.0, 6.0, 15.0);

glTranslated(8.0, 15.0, 0.0);

cylinder(6.0, 6.0, 15.0);

glTranslated(12.0, 15.0, 0.0);

cylinder(6.0, 6.0, 15.0);

glTranslated(16.0, 15.0, 0.0);

cylinder(6.0, 6.0, 15.0);

glPopMatrix();

//Hiasan Pohon Dipojok

glPushMatrix();

glTranslated(-90.0, 25.0, -90.0);

pohon();

glPopMatrix();

glPushMatrix();

glTranslated(-85.0, 25.0, -70.0);

pohon();

glPopMatrix();

//pohon

glPushMatrix();

glTranslated(-90.0, 25.0, 90.0);

pohon();

glTranslated(30.0, 0.0, 0.0);

pohon();

glTranslated(30.0, 0.0, 0.0);

pohon();

glPopMatrix();

//Muter"an

glPushMatrix();

glTranslated(-90.0, 5.0, 50.0);

muterMuteran();

glPopMatrix();

//pager

//pagar ditengah lurus

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.803921568627451, 0.5215686274509804, 0.2470588235294118);

glRotated(-90.0, 1.0, 0.0, 0.0);

glTranslated(0.0, 95.0, 5.0);

cylinder(6.0, 6.0, 15.0);

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

glTranslated(0.0, -15.0, 0.0);

cylinder(6.0, 6.0, 15.0);

}

glPopMatrix();

//pagar di pinggir

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.803921568627451, 0.5215686274509804, 0.2470588235294118);

glRotated(-90.0, 1.0, 0.0, 0.0);

glTranslated(80.0, -35.0, 5.0);

cylinder(6.0, 6.0, 15.0);

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

glTranslated(0.0, -15.0, 0.0);

cylinder(6.0, 6.0, 15.0);

}

glPopMatrix();

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.803921568627451, 0.5215686274509804, 0.2470588235294118);

glRotated(-90.0, 1.0, 0.0, 0.0);

glTranslated(0.0, -10.0, 5.0);

cylinder(6.0, 6.0, 15.0);

for (int x = 0; x < 6; x++) {

glTranslated(-15.0, 0.0, 0.0);

cylinder(6.0, 6.0, 15.0);

}

glPopMatrix();

//Hiasan Pohon Dipojok

glPushMatrix();

glTranslated(-90.0, 25.0, -90.0);

pohon();

glPopMatrix();

glPushMatrix();

glTranslated(-85.0, 25.0, -70.0);

pohon();

glPopMatrix();

glPushMatrix();

glTranslated(-70.0, 20.0, -90.0);

pohon();

glPopMatrix();

//Tempat Minum

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.1, 1.0, 1.0);

glTranslated(-60.0, 10.0, -60.0);

glRotated(-270.0, 1.0, 0.0, 0.0);

cylinder(15.0, 10.0, 3.0);

glPopMatrix();

//tiang 1

////// //tiang 1

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3d(0.0, 0.0, 0.0);

glRotated(-90, 1.0, 0.0, 0.0);

glTranslated(90.0, 100.0, 0.0);

cylinder(3.0, 3.0, 100);

glPopMatrix();

//tiang 2

glPushMatrix();

glRotated(-90, 1.0, 0.0, 0.0);

glTranslated(90.0, -100.0, 0.0);

cylinder(3.0, 3.0, 100);

glPopMatrix();

//tiang 3

glPushMatrix();

glRotated(-90, 1.0, 0.0, 0.0);

glTranslated(-100.0, 100.0, 0.0);

cylinder(3.0, 3.0, 100);

glPopMatrix();

//tiang 4

glPushMatrix();

glRotated(-90, 1.0, 0.0, 0.0);

glTranslated(-100.0, -100.0, 0.0);

cylinder(3.0, 3.0, 100);

glPopMatrix();

// //Bawah

glPushMatrix();

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

glColor3f(0, 1.0, 0.0);

bawahKandang(10);

glPopMatrix();

//atas

// //atas

glPushMatrix();

glTranslated(0.0, 100.0, 0.0);

bawahKandang(10);

glPopMatrix();

glutSwapBuffers();

glFlush();

rot++;

angle++;

}

void init(void) {

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glDepthFunc(GL\_LESS);

glEnable(GL\_NORMALIZE);

glEnable(GL\_COLOR\_MATERIAL);

glDepthFunc(GL\_LEQUAL);

glShadeModel(GL\_SMOOTH);

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

glEnable(GL\_CULL\_FACE);

}

static void kibor(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_HOME:

viewy++;

break;

case GLUT\_KEY\_END:

viewy--;

break;

case GLUT\_KEY\_UP:

viewz--;

break;

case GLUT\_KEY\_DOWN:

viewz++;

break;

case GLUT\_KEY\_RIGHT:

viewx++;

break;

case GLUT\_KEY\_LEFT:

viewx--;

break;

case GLUT\_KEY\_F1: {

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

}

;

break;

case GLUT\_KEY\_F2: {

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient2);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse2);

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

}

;

break;

default:

break;

}

}

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

if (key == 'd') {

spin = spin - 1;

if (spin > 360.0)

spin = spin - 360.0;

}

if (key == 'a') {

spin = spin + 1;

if (spin > 360.0)

spin = spin - 360.0;

}

if (key == 'q') {

viewz++;

}

if (key == 'e') {

viewz--;

}

if (key == 's') {

viewy--;

}

if (key == 'w') {

viewy++;

}

}

void reshape(int w, int h) {

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60, (GLfloat) w / (GLfloat) h, 0.1, 1000.0);

glMatrixMode(GL\_MODELVIEW);

}

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

glutInit(&argc, argv);

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

glutInitWindowSize(800, 600);

glutInitWindowPosition(100, 100);

glutCreateWindow("Sepertinya kandang hamster");

init();

glutDisplayFunc(display);

glutIdleFunc(display);

glutReshapeFunc(reshape);

glutSpecialFunc(kibor);

glutKeyboardFunc(keyboard);

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

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

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

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

glColorMaterial(GL\_FRONT, GL\_DIFFUSE);

glutMainLoop();

return 0;

}

* vec3f.cpp

**#include** <math.h>

**#include** "vec3f.h"

**using** **namespace** std;

**Vec3f::Vec3f**() {

}

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

v[0] = x;

v[1] = y;

v[2] = z;

}

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

**return** v[index];

}

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

**return** v[index];

}

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

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

}

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

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

}

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

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

}

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

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

}

Vec3f **Vec3f::operator-**() **const** {

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

}

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

v[0] \*= scale;

v[1] \*= scale;

v[2] \*= scale;

**return** \***this**;

}

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

v[0] /= scale;

v[1] /= scale;

v[2] /= scale;

**return** \***this**;

}

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

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

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

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

**return** \***this**;

}

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

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

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

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

**return** \***this**;

}

**float** **Vec3f::magnitude**() **const** {

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

}

**float** **Vec3f::magnitudeSquared**() **const** {

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

}

Vec3f **Vec3f::normalize**() **const** {

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

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

}

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

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

}

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

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

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

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

}

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

**return** v \* scale;

}

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

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

**return** output;

}

* imageloader.cpp

**#include** <assert.h>

**#include** <fstream>

**#include** "imageloader.h"

**using** **namespace** std;

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

}

**Image::~Image**() {

**delete**[] pixels;

}

**namespace** {

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

**int** **toInt**(**const** **char**\* bytes) {

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

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

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

(**unsigned** **char**)bytes[0]);

}

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

**short** **toShort**(**const** **char**\* bytes) {

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

(**unsigned** **char**)bytes[0]);

}

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

**int** **readInt**(ifstream &input) {

**char** buffer[4];

input.read(buffer, 4);

**return** toInt(buffer);

}

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

**short** **readShort**(ifstream &input) {

**char** buffer[2];

input.read(buffer, 2);

**return** toShort(buffer);

}

//Just like auto\_ptr, but for arrays

**template**<**class** **T**>

**class** auto\_array {

**private**:

**T**\* array;

**mutable** **bool** isReleased;

**public**:

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

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

}

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

array = aarray.array;

isReleased = aarray.isReleased;

aarray.isReleased = **true**;

}

**~auto\_array**() {

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

**delete**[] array;

}

}

**T**\* **get**() **const** {

**return** array;

}

**T** &**operator\***() **const** {

**return** \*array;

}

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

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

**delete**[] array;

}

array = aarray.array;

isReleased = aarray.isReleased;

aarray.isReleased = **true**;

}

**T**\* **operator->**() **const** {

**return** array;

}

**T**\* **release**() {

isReleased = **true**;

**return** array;

}

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

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

**delete**[] array;

}

array = array\_;

}

**T**\* **operator+**(**int** i) {

**return** array + i;

}

**T** &**operator[]**(**int** i) {

**return** array[i];

}

};

}

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

ifstream input;

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

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

**char** buffer[2];

input.read(buffer, 2);

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

input.ignore(8);

**int** dataOffset = readInt(input);

//Read the header

**int** headerSize = readInt(input);

**int** width;

**int** height;

**switch**(headerSize) {

**case** 40:

//V3

width = readInt(input);

height = readInt(input);

input.ignore(2);

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

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

**break**;

**case** 12:

//OS/2 V1

width = readShort(input);

height = readShort(input);

input.ignore(2);

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

**break**;

**case** 64:

//OS/2 V2

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

**break**;

**case** 108:

//Windows V4

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

**break**;

**case** 124:

//Windows V5

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

**break**;

**default**:

assert(!"Unknown bitmap format");

}

//Read the data

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

**int** size = bytesPerRow \* height;

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

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

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

//Get the data into the right format

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

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

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

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

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

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

}

}

}

input.close();

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

}