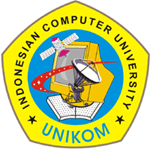
**APLIKASI 3D KINCIR BELANDA**

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

**Disusun oleh:**

**Nurul Al Hadi 10109932**

**Eka Chandra Septiana 10109909**

****

**JURUSAN TEKNIK INFORMATIKA  
FAKULTAS TEKNIK DAN ILMU KOMPUTER  
UNIVERSITAS KOMPUTER INDONESIA  
BANDUNG  
2013**

**BAB I**

* 1. **Latar Belakang**

Belanda tidak hanya terkenal dengan bunga tulip tetapi juga kincir angin. Kincir angin merupakan warisan budaya yang memesona bangsa-bangsa lain sehingga menjadi ikon Belanda dengan sebutan Negeri Kincir Angin.

Kincir angin pada awal keberadaannya di Belanda sekitar abad 13 berfungsi untuk mendorong air ke lautan agar terbentuk daratan baru yang lebih luas (polder). Hal ini mengingat letak dataran Belanda yang sebagian besar wilayahnya berada di bawah permukaan laut.

Dengan perkembangan teknologi, sekitar abad ke-17 kincir angin digunakan juga sebagai sarana pembantu di bidang pertanian dan industri. Seperti memproduksi kertas, mengasah kayu, mengeluarkan minyak dari biji, dan menggiling jagung.

Pada tugas mata kuliah Komputer Grafika kali ini, kami akan membuat pemandangan kincir angin Belanda yang terdiri dari objek kincir angin, bunga tulip, danau, pohon, kandang sapi, pagar.

* 1. **Objek List**

Objek 3D yang terdapat di dalam kandang hamster antara lain:

1. Kincir Angin

Kincir angin yang dibuat akan memiliki animasi putar. Sehingga baling-balingnya akan berputar.

1. Bunga Tulip
2. Danau
3. Pohon
4. Awan
   1. **Objek Pembangun**

Objek pembangun untuk membangun aplikasi ini antara lain:

1. Kincir Angin

Objek pembangun: Balok, silinder, kerucut.

1. Bunga Tulip

Objek pembangun: Bola, balok

1. Danau

Objek pembangun: Terrain.

1. Pohon

Objek pembangun: Silinder, limas.

1. Awan

Objek pembangun: bola.

* 1. **Teknologi**

Aplikasi ini dibangun menggunakan bahasa pemrograman C++ dengan library Open GL, dan tools Eclipse.

**BAB II**

**PEMBAGIAN TUGAS**

**2.1 Pembagian Tugas Anggota Kelompok**

Pada bab ini akan dituliskan tentang pembagian tugas dari masing-masing anggota dari kelompok kami dalam kontribusinya untuk membangun aplikasi 3D Kincir Angin Belanda dengan menggunakan OpenGL. Adapun pembagian tugasnya adalah berdasarkan objek-objek yang akan kami bangun dan diperlukan untuk membangun aplikasi 3D Kincir Angin Belanda tersebut. Berikut adalah daftar pembagian tugas:

**Nurul Al Hadi (10109932)**

**Objek Terrain**

Dibuat dengan gambar .bmp sebagai dasar pembuatan terrain.

**Objek Pohon**

Bagaian batang menggunakan fungsi gluCylinder (pObj, 4, 0.7, 30, 25, 25)

Bagian ranting menggunakan fungsi gluCylinder(pObj, 0.6, 0.1, 15, 25, 25)

Bagian daun menggunakan fungsi glutSolidDodecahedron().

Pohon dibuat sebanyak 5buah

**Animasi putar Kincir Angin**

Animasi ini dibuat menggunakan void timer(int value)

{

putarx +=30;

putary +=4;

glutPostRedisplay();

glutTimerFunc(25,timer,0);

}

Menambah bunga Tulip

Dengan fungsi:

glTranslatef(10-x,10,-6);

if ((i>10)&&(i<15))

{}

else

{bungakuning();}

glPopMatrix();

x +=4;

}

**Menambahkan texture untuk kincir angin**

Menggunakan fungsi:

glEnable(GL\_TEXTURE\_2D);

void freetexture(GLuint texture) {

glDeleteTextures(2, &texture);

}

Untuk mengambil tekstur:

Images \* loadTexture() {

Images \*image1;

image1 = (Images \*) malloc(sizeof(Images));

if (image1 == NULL) {

printf("Error allocating space for image");

exit(0);

}

//index.bmp is a 64x64 picture

if (!ImageLoad("index.bmp", image1)) {

exit(1);

}

return image1;

}

glBindTexture(GL\_TEXTURE\_2D, texture[0]);

**Eka Chandra Septiana (10109909)**

**Objek Bunga Tulip**

Bagian kelopak bungga menggunakan fungsi glutSolidSphere(1.5,100,100); Kelopak dibuat sebanyak 3 buah.

Bagian tangkai bungga menggunakan fungsi gluCylinder(pObj, 0.35, 0.35, 5, 100, 15).

Bunga dibuat beberapa buah, dengan warna yang berbeda-beda.

**Objek Kincir Angin**

Dibuat menggunakan fungsi gluCylinder(pObj, 12, 10, 30, 100, 15), untuk bagian badan kincir, fungsi glutSolidCone(12, 10, 25, 100) untuk bagian atas kincir, fungsi glutSolidCube(2) untuk baling-balingnya.

**Objek Awan**

Dibuat dengan menggunakan fungsi glutSolidSphere(10, 50, 50) sebanyak 5 buah untuk membentuk awan.

**BAB III**

**CODING DAN SCREEN SHOOT**

**3.1 Coding**

/\*

\* Nurul Al Hadi

\* Eka Chandra Septiana

\*

\*

\*/

#include <windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#ifdef \_\_APPLE\_\_

#include <OpenGL/OpenGL.h>

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#include <GL/glu.h>

#include <GL/gl.h>

#include "imageloader.h"

#include "vec3f.h"

#endif

/\*

komen coba commit

\*/

static GLfloat spin, spin2 = 0.0;

float angle = 0;

using namespace std;

GLuint texture[2]; //array untuk texture

//GLint slices = 16;

//GLint stacks = 16;

struct Images {

unsigned long sizeX;

unsigned long sizeY;

char \*data;

};

typedef struct Images Images;

float lastx, lasty;

GLint stencilBits;

static int viewx = -200;

static int viewy = 100;

static int viewz = 200;

float rot = 0;

//class untuk terain 2D

class Terrain {

private:

int w; //Width

int l; //Length

float\*\* hs; //Heights

Vec3f\*\* normals;

bool computedNormals; //Whether normals is up-to-date

public:

Terrain(int w2, int l2) {

w = w2;

l = l2;

hs = new float\*[l];

for (int i = 0; i < l; i++) {

hs[i] = new float[w];

}

normals = new Vec3f\*[l];

for (int i = 0; i < l; i++) {

normals[i] = new Vec3f[w];

}

computedNormals = false;

}

~Terrain() {

for (int i = 0; i < l; i++) {

delete[] hs[i];

}

delete[] hs;

for (int i = 0; i < l; i++) {

delete[] normals[i];

}

delete[] normals;

}

int width() {

return w;

}

int length() {

return l;

}

//Sets the height at (x, z) to y

void setHeight(int x, int z, float y) {

hs[z][x] = y;

computedNormals = false;

}

//Returns the height at (x, z)

float getHeight(int x, int z) {

return hs[z][x];

}

//Computes the normals, if they haven't been computed yet

void computeNormals() {

if (computedNormals) {

return;

}

//Compute the rough version of the normals

Vec3f\*\* normals2 = new Vec3f\*[l];

for (int i = 0; i < l; i++) {

normals2[i] = new Vec3f[w];

}

for (int z = 0; z < l; z++) {

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

Vec3f sum(0.0f, 0.0f, 0.0f);

Vec3f out;

if (z > 0) {

out = Vec3f(0.0f, hs[z - 1][x] - hs[z][x], -1.0f);

}

Vec3f in;

if (z < l - 1) {

in = Vec3f(0.0f, hs[z + 1][x] - hs[z][x], 1.0f);

}

Vec3f left;

if (x > 0) {

left = Vec3f(-1.0f, hs[z][x - 1] - hs[z][x], 0.0f);

}

Vec3f right;

if (x < w - 1) {

right = Vec3f(1.0f, hs[z][x + 1] - hs[z][x], 0.0f);

}

if (x > 0 && z > 0) {

sum += out.cross(left).normalize();

}

if (x > 0 && z < l - 1) {

sum += left.cross(in).normalize();

}

if (x < w - 1 && z < l - 1) {

sum += in.cross(right).normalize();

}

if (x < w - 1 && z > 0) {

sum += right.cross(out).normalize();

}

normals2[z][x] = sum;

}

}

//Smooth out the normals

const float FALLOUT\_RATIO = 0.5f;

for (int z = 0; z < l; z++) {

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

Vec3f sum = normals2[z][x];

if (x > 0) {

sum += normals2[z][x - 1] \* FALLOUT\_RATIO;

}

if (x < w - 1) {

sum += normals2[z][x + 1] \* FALLOUT\_RATIO;

}

if (z > 0) {

sum += normals2[z - 1][x] \* FALLOUT\_RATIO;

}

if (z < l - 1) {

sum += normals2[z + 1][x] \* FALLOUT\_RATIO;

}

if (sum.magnitude() == 0) {

sum = Vec3f(0.0f, 1.0f, 0.0f);

}

normals[z][x] = sum;

}

}

for (int i = 0; i < l; i++) {

delete[] normals2[i];

}

delete[] normals2;

computedNormals = true;

}

//Returns the normal at (x, z)

Vec3f getNormal(int x, int z) {

if (!computedNormals) {

computeNormals();

}

return normals[z][x];

}

};

//end class

//Loads a terrain from a heightmap. The heights of the terrain range from

//-height / 2 to height / 2.

Terrain\* loadTerrain(const char\* filename, float height) {

Image\* image = loadBMP(filename);

Terrain\* t = new Terrain(image->width, image->height);

for (int y = 0; y < image->height; y++) {

for (int x = 0; x < image->width; x++) {

unsigned char color = (unsigned char) image->pixels[3 \* (y

\* image->width + x)];

float h = height \* ((color / 255.0f) - 0.5f);

t->setHeight(x, y, h);

}

}

delete image;

t->computeNormals();

return t;

}

float \_angle = 45.0f;

Terrain\* \_terrainBukit;

Terrain\* \_terrainAir;

void cleanup() {

delete \_terrainBukit;

delete \_terrainAir;

}

void initRendering() {

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_NORMALIZE);

glShadeModel(GL\_SMOOTH);

glEnable(GL\_TEXTURE\_2D);

}

void drawScene() {

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

float scale = 500.0f / max(\_terrainBukit->width() - 1, \_terrainBukit->length() - 1);

glScalef(scale, scale, scale);

glTranslatef(-(float) (\_terrainBukit->width() - 1) / 2, 0.0f,

-(float) (\_terrainBukit->length() - 1) / 2);

glColor3f(0.3f, 0.9f, 0.0f);

for (int z = 0; z < \_terrainBukit->length() - 1; z++) {

//Makes OpenGL draw a triangle at every three consecutive vertices

glBegin(GL\_TRIANGLE\_STRIP);

for (int x = 0; x < \_terrainBukit->width(); x++) {

Vec3f normal = \_terrainBukit->getNormal(x, z);

glNormal3f(normal[0], normal[1], normal[2]);

glVertex3f(x, \_terrainBukit->getHeight(x, z), z);

normal = \_terrainBukit->getNormal(x, z + 1);

glNormal3f(normal[0], normal[1], normal[2]);

glVertex3f(x, \_terrainBukit->getHeight(x, z + 1), z + 1);

}

glEnd();

}

}

void gambarTanah(Terrain \*terrain, GLfloat r, GLfloat g, GLfloat b) {

float scale = 400.0f / max(terrain->width() - 1, terrain->length() - 1);

glScalef(scale, scale, scale);

glTranslatef(-(float) (terrain->width() - 1) / 2, 0.0f,

-(float) (terrain->length() - 1) / 2);

glColor3f(r, g, b);

for (int z = 0; z < terrain->length() - 1; z++) {

//Makes OpenGL draw a triangle at every three consecutive vertices

glBegin(GL\_TRIANGLE\_STRIP);

for (int x = 0; x < terrain->width(); x++) {

Vec3f normal = terrain->getNormal(x, z);

glNormal3f(normal[0], normal[1], normal[2]);

glVertex3f(x, terrain->getHeight(x, z), z);

normal = terrain->getNormal(x, z + 1);

glNormal3f(normal[0], normal[1], normal[2]);

glVertex3f(x, terrain->getHeight(x, z + 1), z + 1);

}

glEnd();

}

}

void gambarAir(Terrain \*terrain, GLfloat r, GLfloat g, GLfloat b) {

float scale = 350.0f / max(terrain->width() - 1, terrain->length() - 1);

glScalef(scale, scale, scale);

glTranslatef(-(float) (terrain->width() - 1) / 2, 0.0f,

-(float) (terrain->length() - 1) / 2);

glColor3f(r, g, b);

for (int z = 0; z < terrain->length() - 1; z++) {

//Makes OpenGL draw a triangle at every three consecutive vertices

glBegin(GL\_TRIANGLE\_STRIP);

for (int x = 0; x < terrain->width(); x++) {

Vec3f normal = terrain->getNormal(x, z);

glNormal3f(normal[0], normal[1], normal[2]);

glVertex3f(x, terrain->getHeight(x, z), z);

normal = terrain->getNormal(x, z + 1);

glNormal3f(normal[0], normal[1], normal[2]);

glVertex3f(x, terrain->getHeight(x, z + 1), z + 1);

}

glEnd();

}

}

void update(int value) {

glutPostRedisplay();

glutTimerFunc(25, update, 0);

}

void freetexture(GLuint texture) {

glDeleteTextures(2, &texture);

}

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 pohon(void){

//batang

GLUquadricObj \*pObj;

pObj =gluNewQuadric();

gluQuadricNormals(pObj, GLU\_SMOOTH);

glPushMatrix();

glColor3ub(104,70,14);

glRotatef(270,1,0,0);

gluCylinder(pObj, 4, 0.7, 30, 25, 25);

glPopMatrix();

}

//ranting

void ranting(void){

GLUquadricObj \*pObj;

pObj =gluNewQuadric();

gluQuadricNormals(pObj, GLU\_SMOOTH);

glPushMatrix();

glColor3ub(104,70,14);

glTranslatef(0,27,0);

glRotatef(330,1,0,0);

gluCylinder(pObj, 0.6, 0.1, 15, 25, 25);

glPopMatrix();

//daun

glPushMatrix();

glColor3ub(18,118,13);

glScaled(5, 5, 5);

glTranslatef(0,7,3);

glutSolidDodecahedron();

glPopMatrix();

}

//bunga

void bungamerah()

{

GLUquadricObj \*pObj;

pObj =gluNewQuadric();

gluQuadricNormals(pObj, GLU\_SMOOTH);

glColor3f(1,0,0);

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

glPushMatrix();

glTranslatef(-4, 0, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(1,0,0);

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

glPushMatrix();

glTranslatef(-5, 0, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(1,0,0);

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

glPushMatrix();

glTranslatef(-4.5, 1, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(0.3,0.2,0);

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

glPushMatrix();

glTranslatef(-4, -5, 0);

glRotatef(270,1,0,0);

gluCylinder(pObj, 0.35, 0.35, 5, 100, 15); //ptr, rbase, rtop, height, slices, stacks

glPopMatrix();

}

void bungaungu()

{

GLUquadricObj \*pObj;

pObj =gluNewQuadric();

gluQuadricNormals(pObj, GLU\_SMOOTH);

glColor3f(0.2745,0.0039,0.2745);

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

glPushMatrix();

glTranslatef(-4, 0, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(0.2745,0.0039,0.2745);

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

glPushMatrix();

glTranslatef(-5, 0, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(0.2745,0.0039,0.2745);

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

glPushMatrix();

glTranslatef(-4.5, 1, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(0.3,0.2,0);

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

glPushMatrix();

glTranslatef(-4, -5, 0);

glRotatef(270,1,0,0);

gluCylinder(pObj, 0.35, 0.35, 5, 100, 15); //ptr, rbase, rtop, height, slices, stacks

glPopMatrix();

}

void bungapink()

{

GLUquadricObj \*pObj;

pObj =gluNewQuadric();

gluQuadricNormals(pObj, GLU\_SMOOTH);

glColor3f(0.976,0.604,0.961);

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

glPushMatrix();

glTranslatef(-4, 0, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(0.976,0.604,0.961);

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

glPushMatrix();

glTranslatef(-5, 0, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(0.976,0.604,0.961);

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

glPushMatrix();

glTranslatef(-4.5, 1, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(0.3,0.2,0);

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

glPushMatrix();

glTranslatef(-4, -5, 0);

glRotatef(270,1,0,0);

gluCylinder(pObj, 0.35, 0.35, 5, 100, 15); //ptr, rbase, rtop, height, slices, stacks

glPopMatrix();

}

//bunga kuning

void bungakuning()

{

GLUquadricObj \*pObj;

pObj =gluNewQuadric();

gluQuadricNormals(pObj, GLU\_SMOOTH);

glColor3f(0.929,0.988,0.008);

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

glPushMatrix();

glTranslatef(-4, 0, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(0.929,0.988,0.008);

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

glPushMatrix();

glTranslatef(-5, 0, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(0.929,0.988,0.008);

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

glPushMatrix();

glTranslatef(-4.5, 1, 0);

glutSolidSphere(1.5,100,100);

glPopMatrix();

glColor3f(0.3,0.2,0);

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

glPushMatrix();

glTranslatef(-4, -5, 0);

glRotatef(270,1,0,0);

gluCylinder(pObj, 0.35, 0.35, 5, 100, 15); //ptr, rbase, rtop, height, slices, stacks

glPopMatrix();

}

//tambahan dari ekok

//kincir angin

static int putarx=90;

static int putary=0;

void kincir()

{

GLUquadricObj \*pObj;

pObj =gluNewQuadric();

gluQuadricNormals(pObj, GLU\_SMOOTH);

//badan

glColor3f(0.7f,1.0f,0.7f);

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

glPushMatrix();

glTranslatef(-4, -5, 0);

glRotatef(270,1,0,0);

gluCylinder(pObj, 12, 10, 30, 100, 15); //ptr, rbase, rtop, height, slices, stacks

glPopMatrix();

//atas

glColor3f(0.3,0.2,0);

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

glPushMatrix();

glTranslatef(-4, 25, 0);

glRotatef(270,1,0,0);

glutSolidCone(12, 10, 25, 100);

glPopMatrix();

}

void baling()

{

glPushMatrix();

glColor3f(0.929,0.988,0.008);

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

glTranslatef(-15, 0, 0);

glRotatef(90,0,0,1);

glRotatef(45,0,1,0);

glScalef(7,0.3,0.5);

glutSolidCube(5);

glPopMatrix();

glPushMatrix();

glColor3f(1,0,0);

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

glTranslatef(-15, 0, 0);

glRotatef(90,0,0,1);

glRotatef(45,0,1,0);

glScalef(5,1,3);

glutSolidCube(2);

glPopMatrix();

glPushMatrix();

glColor3f(1,0,0);

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

glTranslatef(-15, 0, 0);

glRotatef(90,0,0,1);

glRotatef(135,0,1,0);

glScalef(5,1,3);

glutSolidCube(2);

glPopMatrix();

//baling2

glPushMatrix();

glColor3f(0.929,0.988,0.008);

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

glTranslatef(-15, 0, 0);

glRotatef(90,0,0,1);

glRotatef(135,0,1,0);

glScalef(7,0.3,0.5);

glutSolidCube(5);

glPopMatrix();

glPushMatrix();

glColor3f(1,0,0);

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

glTranslatef(-15, 0, 0);

glRotatef(90,0,0,1);

glRotatef(135,0,1,0);

glScalef(5,1,3);

glutSolidCube(2);

glPopMatrix();

glPushMatrix();

glColor3f(1,0,0);

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

glTranslatef(-15, 0, 0);

glRotatef(90,0,0,1);

glRotatef(45,0,1,0);

glScalef(5,1,3);

glutSolidCube(2);

glPopMatrix();

}

void timer(int value)

{

//tuliskan varibel yang berubah nilainya disini

putarx +=30;

putary +=4;

glutPostRedisplay();

glutTimerFunc(25,timer,0);

}

int ImageLoad(char \*filename, Images \*image) {

FILE \*file;

unsigned long size; // ukuran image dalam bytes

unsigned long i; // standard counter.

unsigned short int plane; // number of planes in image

unsigned short int bpp; // jumlah bits per pixel

char temp; // temporary color storage for var warna sementara untuk memastikan filenya ada

if ((file = fopen(filename, "rb")) == NULL) {

printf("File Not Found : %s\n", filename);

return 0;

}

// mencari file header bmp

fseek(file, 18, SEEK\_CUR);

// read the width

if ((i = fread(&image->sizeX, 4, 1, file)) != 1) {

printf("Error reading width from %s.\n", filename);

return 0;

}

//printf("Width of %s: %lu\n", filename, image->sizeX);

// membaca nilai height

if ((i = fread(&image->sizeY, 4, 1, file)) != 1) {

printf("Error reading height from %s.\n", filename);

return 0;

}

//printf("Height of %s: %lu\n", filename, image->sizeY);

//menghitung ukuran image(asumsi 24 bits or 3 bytes per pixel).

size = image->sizeX \* image->sizeY \* 3;

// read the planes

if ((fread(&plane, 2, 1, file)) != 1) {

printf("Error reading planes from %s.\n", filename);

return 0;

}

if (plane != 1) {

printf("Planes from %s is not 1: %u\n", filename, plane);

return 0;

}

// read the bitsperpixel

if ((i = fread(&bpp, 2, 1, file)) != 1) {

printf("Error reading bpp from %s.\n", filename);

return 0;

}

if (bpp != 24) {

printf("Bpp from %s is not 24: %u\n", filename, bpp);

return 0;

}

// seek past the rest of the bitmap header.

fseek(file, 24, SEEK\_CUR);

// read the data.

image->data = (char \*) malloc(size);

if (image->data == NULL) {

printf("Error allocating memory for color-corrected image data");

return 0;

}

if ((i = fread(image->data, size, 1, file)) != 1) {

printf("Error reading image data from %s.\n", filename);

return 0;

}

for (i = 0; i < size; i += 3) { // membalikan semuan nilai warna (gbr - > rgb)

temp = image->data[i];

image->data[i] = image->data[i + 2];

image->data[i + 2] = temp;

}

// we're done.

return 1;

}

//mengambil tekstur

Images \* loadTexture() {

Images \*image1;

// alokasi memmory untuk tekstur

image1 = (Images \*) malloc(sizeof(Images));

if (image1 == NULL) {

printf("Error allocating space for image");

exit(0);

}

//index.bmp is a 64x64 picture

if (!ImageLoad("TextureBata.bmp", image1)) {

exit(1);

}

return image1;

}

void awan()

{

glPushMatrix();

glColor3f(1, 1, 1);

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

glutSolidSphere(10, 50, 50);

glPopMatrix();

glPushMatrix();

glTranslatef(10,0,1);

glutSolidSphere(5, 50, 50);

glPopMatrix();

glPushMatrix();

glTranslatef(-2,6,-2);

glutSolidSphere(7, 50, 50);

glPopMatrix();

glPushMatrix();

glTranslatef(-10,-3,0);

glutSolidSphere(7, 50, 50);

glPopMatrix();

glPushMatrix();

glTranslatef(6,-2,2);

glutSolidSphere(7, 50, 50);

glPopMatrix();

}

void display(void){

// glutSwapBuffers();

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);

glLoadIdentity();

gluLookAt(viewx, viewy, viewz, 0.0, 10.0, 0.0, 0.0, 1.0, 0.0);

//gluLookAt(0.0,10.0,3.0,0.0,0.0,0.0,0.0,1.0,0.0);

glPushMatrix();

glPopMatrix();

glPushMatrix();

//glBindTexture(GL\_TEXTURE\_2D, texture[2]);

gambarTanah(\_terrainBukit, 0.3f, 0.9f, 0.0f);

glPopMatrix();

glPushMatrix();

gambarAir(\_terrainAir, 0.0f, 0.2f, 0.5f);

glPopMatrix();

//pohon1

glPushMatrix();

glTranslatef(-80,0,-120);

glRotatef(90,0,1,0);

pohon();

ranting();

glPushMatrix();

glScalef(1.5, 1.5, 1.5);

glTranslatef(0,25,25);

glRotatef(250,1,0,0);

ranting();

glPopMatrix();

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(0,-6,21.5);

glRotatef(-55,1,0,0);

ranting();

glPopMatrix();

glPopMatrix();

//pohon1

glPushMatrix();

glTranslatef(-30,0,-130);

glRotatef(90,0,1,0);

pohon();

ranting();

glPushMatrix();

glScalef(1.5, 1.5, 1.5);

glTranslatef(0,25,25);

glRotatef(250,1,0,0);

ranting();

glPopMatrix();

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(0,-6,21.5);

glRotatef(-55,1,0,0);

ranting();

glPopMatrix();

glPopMatrix();

//p3

glPushMatrix();

glTranslatef(-135,0,-90);

glRotatef(90,0,1,0);

pohon();

ranting();

glPushMatrix();

glScalef(1.5, 1.5, 1.5);

glTranslatef(0,25,25);

glRotatef(250,1,0,0);

ranting();

glPopMatrix();

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(0,-6,21.5);

glRotatef(-55,1,0,0);

ranting();

glPopMatrix();

glPopMatrix();

//pohon4

glPushMatrix();

glTranslatef(-160,0,-60);

glRotatef(90,0,1,0);

pohon();

ranting();

glPushMatrix();

glScalef(1.5, 1.5, 1.5);

glTranslatef(0,25,25);

glRotatef(250,1,0,0);

ranting();

glPopMatrix();

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(0,-6,21.5);

glRotatef(-55,1,0,0);

ranting();

glPopMatrix();

glPopMatrix();

glPopMatrix();

//pohon5

glPushMatrix();

glTranslatef(-180,0,-0);

glRotatef(90,0,1,0);

pohon();

ranting();

glPushMatrix();

glScalef(1.5, 1.5, 1.5);

glTranslatef(0,25,25);

glRotatef(250,1,0,0);

ranting();

glPopMatrix();

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(0,-6,21.5);

glRotatef(-55,1,0,0);

ranting();

glPopMatrix();

glPopMatrix();

glPopMatrix();

int x=0;

for (int i=1;i<23;i++)

{

//bunga merah

glPushMatrix();

glRotatef(90,0,1,0);

glScalef(1.8, 1.8, 1.8);

glTranslatef(10-x,10,-10);

if ((i>10)&&(i<15))

{}

else

{bungamerah();}

glPopMatrix();

x +=4;

}

x=0;

for (int i=1;i<23;i++)

{

//bunga ungu

glPushMatrix();

glRotatef(90,0,1,0);

glScalef(1.8, 1.8, 1.8);

glTranslatef(10-x,10,-2);

if ((i>10)&&(i<15))

{}

else

{bungapink();}

glPopMatrix();

x +=4;

}

x=0;

for (int i=1;i<23;i++)

{

//bunga kuning

glPushMatrix();

glRotatef(90,0,1,0);

glScalef(1.8, 1.8, 1.8);

glTranslatef(10-x,10,-6);

if ((i>10)&&(i<15))

{}

else

{bungakuning();}

glPopMatrix();

x +=4;

}

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(10,65,-30);

awan();

glPopMatrix();

//awan2

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(50,65,-30);

awan();

glPopMatrix();

//awan3

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glRotated(45,1,0,0);

glTranslatef(-50,65,-30);

awan();

glPopMatrix();

//awan4

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glRotated(45,1,0,0);

glTranslatef(30,65,-30);

awan();

glPopMatrix();

//tambahan manggil kincir angin

//kincir1

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(50,10,-30);

glBindTexture(GL\_TEXTURE\_2D, texture[0]);

kincir();

glPopMatrix();

//kincir2

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(50,10,50);

kincir();

glPopMatrix();

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(45,30,55);

glRotated(putarx,1,0,0);

baling();

glPopMatrix();

//baling-baling

glPushMatrix();

glScalef(1.8, 1.8, 1.8);

glTranslatef(40,30,-20);

glRotated(putarx,1,0,0);

baling();

glPopMatrix();

glColorMask(GL\_FALSE, GL\_FALSE, GL\_FALSE, GL\_FALSE); //disable the color mask

glDepthMask(GL\_FALSE); //disable the depth mask

glEnable(GL\_STENCIL\_TEST); //enable the stencil testing

glStencilFunc(GL\_ALWAYS, 1, 0xFFFFFFFF);

glStencilOp(GL\_REPLACE, GL\_REPLACE, GL\_REPLACE); //set the stencil buffer to replace our next lot of data

//ground

//tanah(); //set the data plane to be replaced

glColorMask(GL\_TRUE, GL\_TRUE, GL\_TRUE, GL\_TRUE); //enable the color mask

glDepthMask(GL\_TRUE); //enable the depth mask

glStencilFunc(GL\_EQUAL, 1, 0xFFFFFFFF);

glStencilOp(GL\_KEEP, GL\_KEEP, GL\_KEEP); //set the stencil buffer to keep our next lot of data

glDisable(GL\_DEPTH\_TEST); //disable depth testing of the reflection

// glPopMatrix();

glEnable(GL\_DEPTH\_TEST); //enable the depth testing

glDisable(GL\_STENCIL\_TEST); //disable the stencil testing

//end of ground

glEnable(GL\_BLEND); //enable alpha blending

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA); //set the blending function

glRotated(1, 0, 0, 0);

glDisable(GL\_BLEND);

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);

glEnable(GL\_TEXTURE\_2D);

glEnable(GL\_TEXTURE\_GEN\_S);

glEnable(GL\_TEXTURE\_GEN\_T);

initRendering();

\_terrainBukit = loadTerrain("TerrainBukit.bmp", 8);

\_terrainAir = loadTerrain("TerrainAir.bmp",0);

Images \*image1 = loadTexture();

if (image1 == NULL) {

printf("Image was not returned from loadTexture\n");

exit(0);

}

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

// Generate texture/ membuat texture

glGenTextures(2, texture);

glBindTexture(GL\_TEXTURE\_2D, texture[0]);

//menyesuaikan ukuran textur ketika image lebih besar dari texture

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

//menyesuaikan ukuran textur ketika image lebih kecil dari texture

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

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, image1->sizeX, image1->sizeY, 0, GL\_RGB,

GL\_UNSIGNED\_BYTE, image1->data);

//baris tekstur buatan #belang

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

}

void reshape(int w, int h){

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45, (GLfloat) w / (GLfloat) h, 0.1, 1000.0);

glMatrixMode(GL\_MODELVIEW);

}

static void keyboard(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++;

}

}

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("Taman Kincir Belanda");

init();

glutDisplayFunc(display);

glutIdleFunc(display);

glutReshapeFunc(reshape);

glutTimerFunc(500, timer, 0);

glutKeyboardFunc (keyboard);

glutSpecialFunc(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);

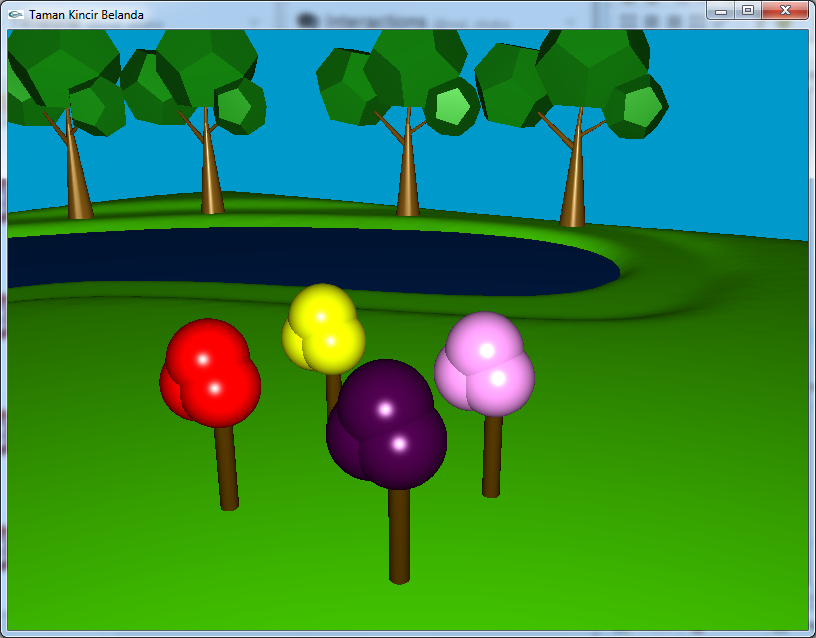
glutMainLoop();

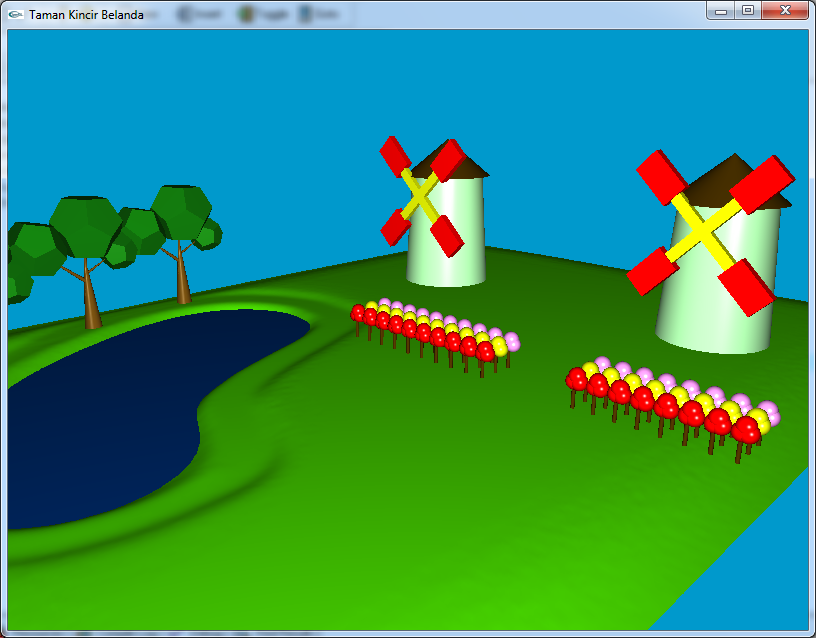
return 0;

}

**3.2 Screen Shoot**

**a. Objek Bunga Tulip**

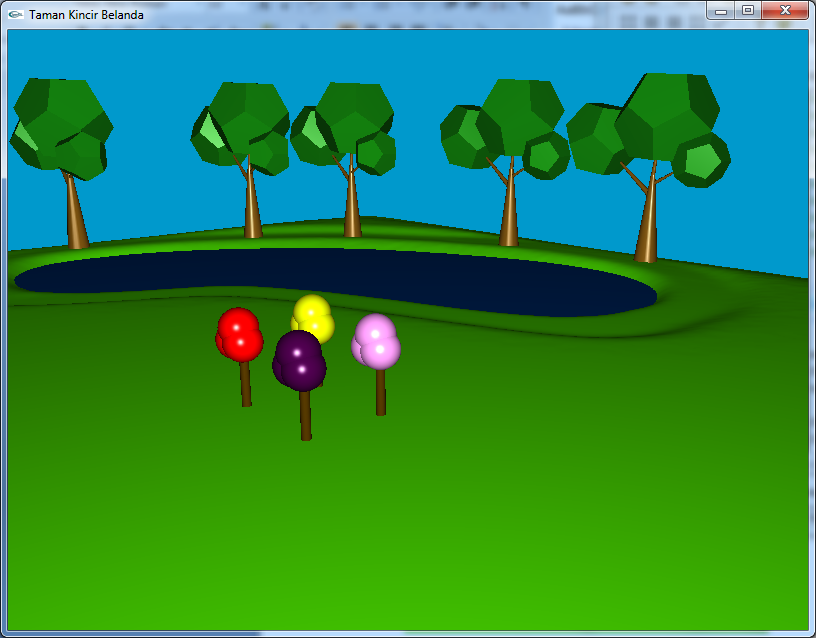




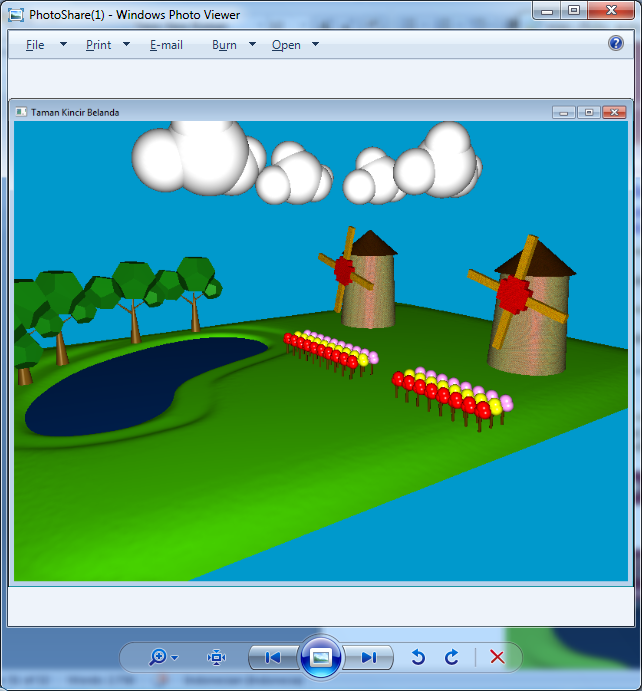
b. Objek Danau



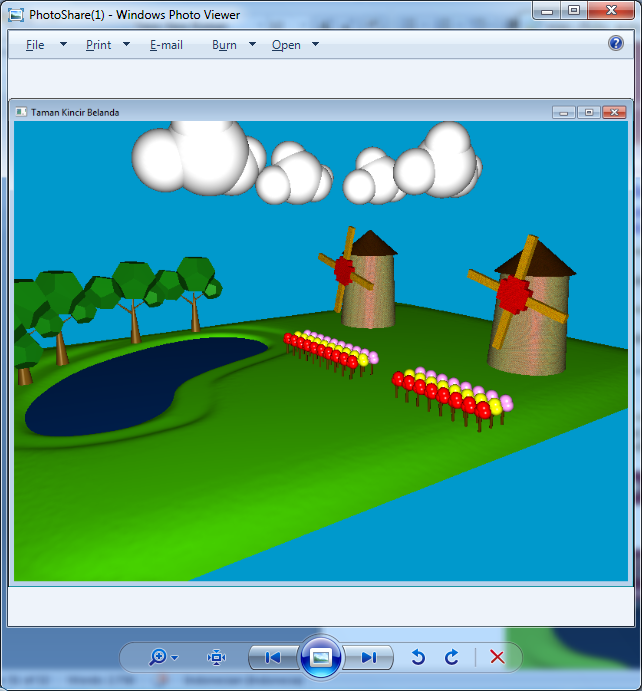
c. Objek Pohon



d. Objek Kincir Angin



e. Objek Awan



f. Tampilan Awal pada saat di run:

