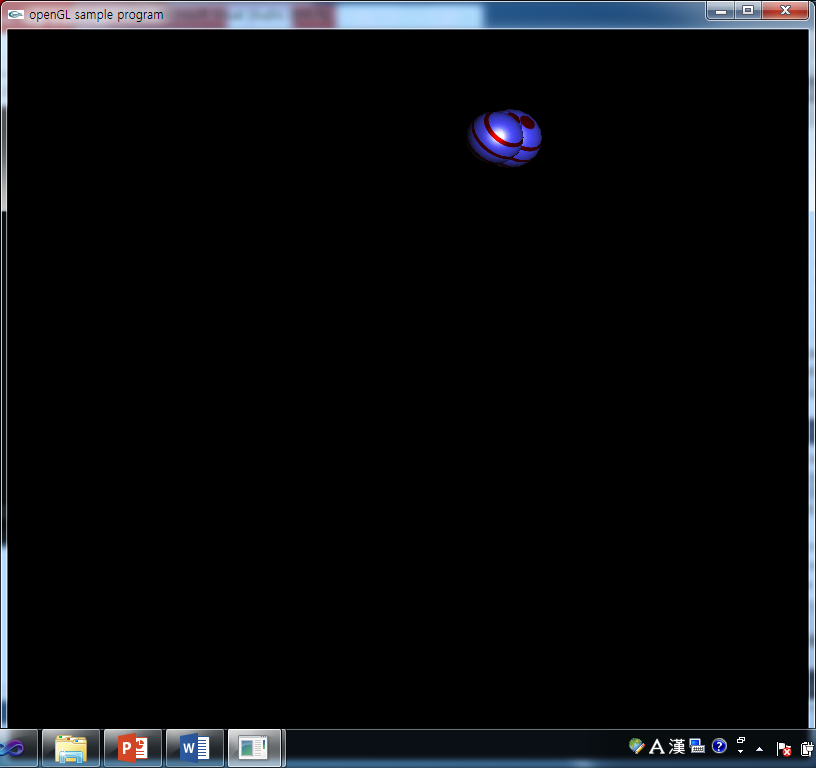
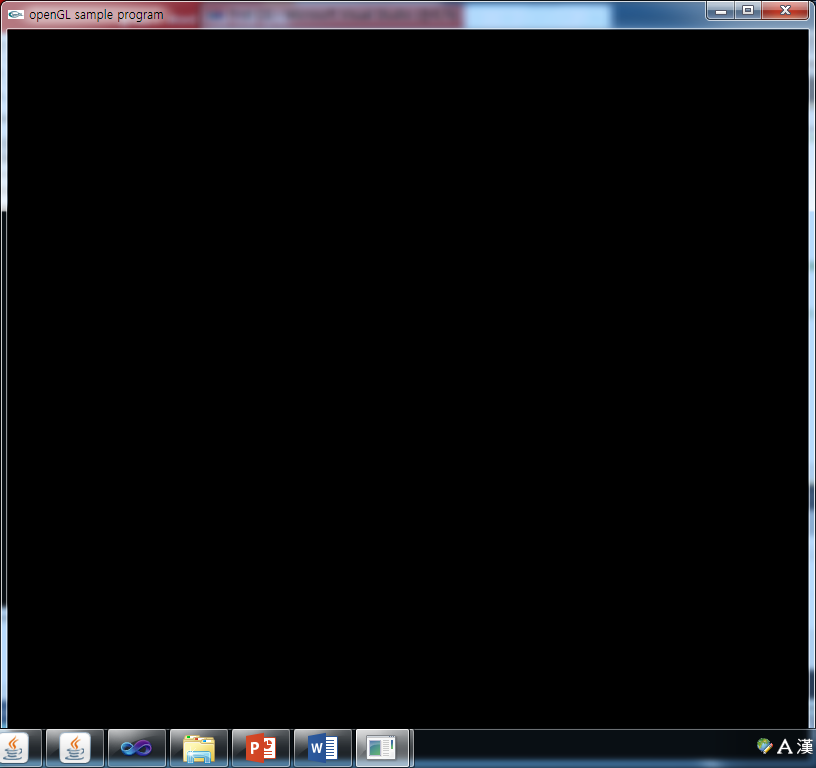
**HW 5. Particle System III**

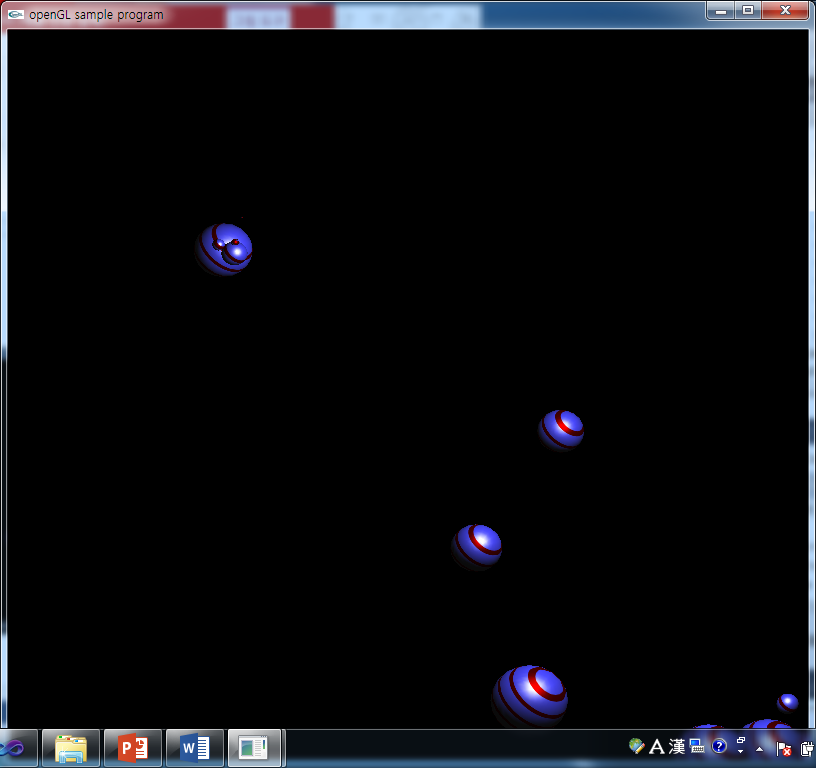
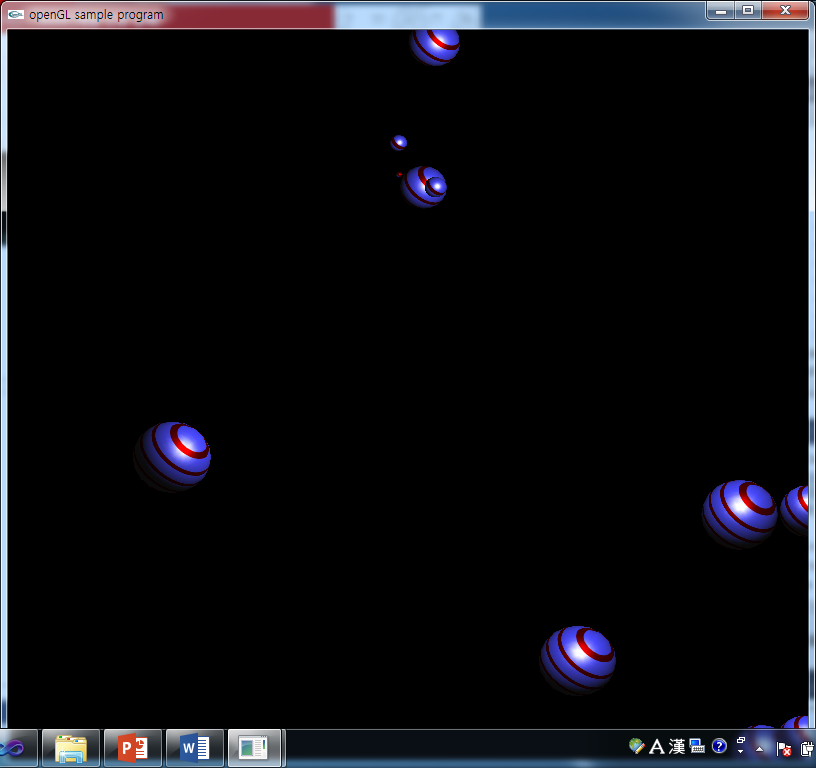
**201133216**

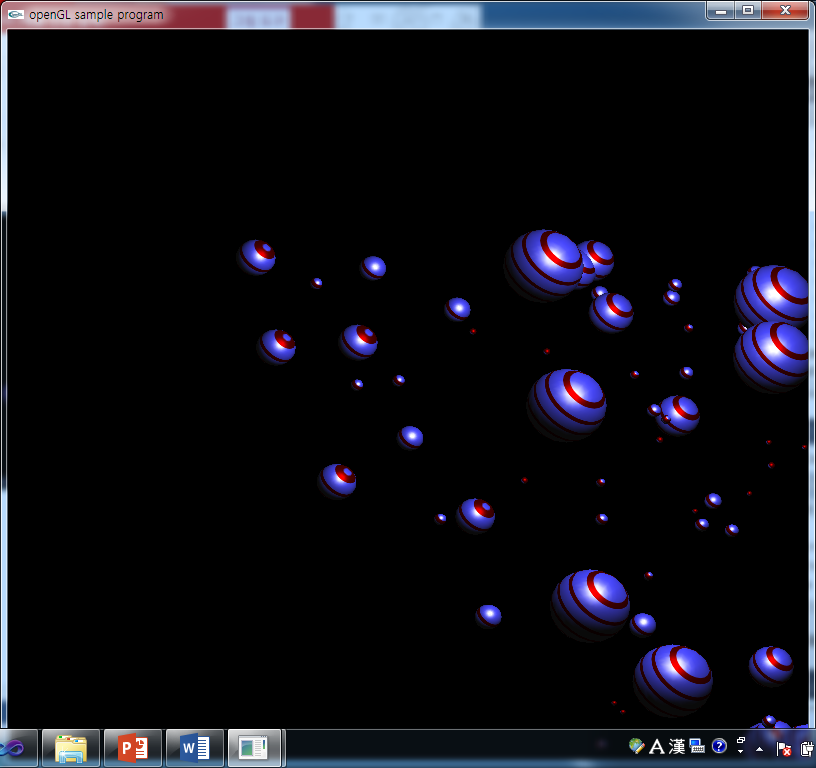
**정유석**

* Each time you click the mouse, a new particle sy.stem is created at the mouse’s location.
* A particle system has file particles.
* Each particle drops down by gravity

**ScreenShot**







**CODE**

**<Source.cpp>**

#include <stdlib.h>

#include <GL/glut.h>

#include <GL/glu.h>

#include <time.h>

#include "CParticleSystem.h"

#include<stdio.h>

GLfloat mat\_diffuse[ ] = {0.25, 0.25, 1., 0.}; //확¢ç산íe반öY사íc

GLfloat mat\_specular[ ] = {1., 1., 1., 0.}; //경Æ©¡면¬e반öY사íc

GLfloat light\_position[ ] = {10., 10., 20., 1.};//광¾¢´원¯©ª의C 위¡×치¢®

GLfloat ref\_plane[ ] = {1.5, 1.5, 1.5, 0.}; //텍¨ª스ö¨¬쳐A 기¾a준¨ª평¡o면¬e

GLUquadricObj \*qobj; //물ö¡Æ체¨ù 포¡¡À인I터I

unsigned int MyTextureObject; //텍¨ª스ö¨¬쳐A 객Æ¢¥체¨ù면¬e

#define stripeImageWidth 32

#define MAX\_PARTICLES 100

#define MAX\_BOUNCE\_COUNT 5000

#define MAX\_PARTICLE\_AGE 5000

GLubyte stripeImage[4\*stripeImageWidth];//텍¨ª스ö¨¬쳐A 배öe열¯¡©

int delay=10; //Delay of timeer function

void InitParticles(); //Activate particles and define the target speed

void ActivateParticles(); // Set the initial position and set a random speed for each

void AdjustParticles(); // Draw of each particles

void RenderParticles(); //Represent each particle by a ball

int count = 0; //array parameter

ParticleSystem \*PS[100]; //Make each particle from user touch event

//Activate particles and define the target speed

void InitParticles(int num, int posX, int posY){

count++;

//When user click the window, it makes 5 particles

PS[count] = new ParticleSystem(num, posX, posY);

}

void ActivateParticles(){

}

//Update the speed and the position of the particle by the new speed

void AdjustParticles(int i){

PS[i]->Update(0.1);

}

//Represent each particle by a ball

void RenderParticles(int i){

glPushMatrix();

glTranslatef(-10, -11, 0);

PS[i]->Draw(); //Draw particles

glPopMatrix();

}

void MyStripeImage( ) { //텍¨ª스ö¨¬쳐A 생íy성ù¨¬함O수ùo

for (int j = 0; j < stripeImageWidth; j++) {

stripeImage[4\*j] = 255;

stripeImage[4\*j+1] = (j < 8) ? 0 : 255;

stripeImage[4\*j+2] = (j < 8) ? 0 : 255;

stripeImage[4\*j+3] = 0;

}

}

void MyDisplay( ){

//declare our using glut function (depth, color buffer bit)

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

glBindTexture(GL\_TEXTURE\_1D, MyTextureObject);

for(int i=1; i<=count; i++){ //Create particles

//if user click display, this program creates new 5 particles

// preview particles remain our program

AdjustParticles(i);

RenderParticles(i);

}

glutSwapBuffers( ); //Swap depth buffer

}

void Init( ){

qobj = gluNewQuadric( );

gluQuadricDrawStyle(qobj, GLU\_FILL);

//set material part of light

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

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

glMaterialf(GL\_FRONT, GL\_SHININESS, 25.0);

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

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_DEPTH\_TEST);

glShadeModel(GL\_SMOOTH);

//set texture to particles

MyStripeImage( );

glGenTextures(1, &MyTextureObject);

glBindTexture(GL\_TEXTURE\_1D, MyTextureObject);

glTexImage1D(GL\_TEXTURE\_1D, 0, 4, stripeImageWidth, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, stripeImage);

glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);

glTexGenfv(GL\_S, GL\_OBJECT\_PLANE, ref\_plane);

glTexParameterf(GL\_TEXTURE\_1D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameterf(GL\_TEXTURE\_1D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_1D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

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

glEnable(GL\_TEXTURE\_GEN\_S);

glEnable(GL\_TEXTURE\_1D);

//Initial particles (first, It doesn't create particle so it sets 0,0,0)

InitParticles(0,0,0);

ActivateParticles();

}

void timer(int t){ //Timer function (each delay, it runs this program)

glutPostRedisplay();

glutTimerFunc( delay, timer, t );

}

void MyReshape(int w, int h){ //set projection & model view

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION); //set projection

glLoadIdentity( ); //Initialize CMT

glOrtho (-10.0, 10.0, -10.0, 10.0, -20.0, 20.0); //set orthographic

glMatrixMode(GL\_MODELVIEW); //set model view

glLoadIdentity ( ); //initialize CMT

gluLookAt(0., 0., 10., 0., 0., 0., 0., 1., 0.); //set camera

glTranslatef(0., 0., -1.);

}

void myMouseFunc(GLint Button, GLint State, GLint x, GLint y){

int posX, posY; //Store convert coordinate (Screen -> OpenGL)

//Statement of mouse left button down

if(State == GLUT\_DOWN && Button == GLUT\_LEFT\_BUTTON){

posX = x / 40; //Convert screen coordinate to program coordinate

posY = 20 - y / 40; //Because screen coordinate, orthographic coordinate so different coordinate

InitParticles(5,posX,posY); //Set particles in this program

ActivateParticles(); //Particles act

}

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);//Add display mode(RGB, Depth, Double frame buffer)

glutInitWindowSize(800, 800);//Set window size(width, height)

glutCreateWindow("openGL sample program"); //Set window title

glutReshapeFunc(MyReshape); //If window size is changed, 'MyTeshape' function call

glutDisplayFunc(MyDisplay);//Call display function

glutTimerFunc( delay, timer, 0 ); //Timer function

Init( ); //Init this program.

glutMouseFunc(myMouseFunc);//Register mouse event handler to system

glutMainLoop( ); //Main loop

}

**<CParticleSystem.h>**

#include "CParticle.h"

class ParticleSystem {

int NumParticles; //number of particles

Particle \*P;

public:

ParticleSystem(int num, int posX, int posY);

~ParticleSystem();

void Update(float deltaTime);

void Draw();

};

**<CParticleSystem.cpp>**

#include <stdlib.h>

#include "CParticleSystem.h"

#include <time.h>

//(posX, posY are location of initial particle)

ParticleSystem::ParticleSystem(int num, int posX, int posY) {

NumParticles = num; //set number of particle (5)

P = new Particle[NumParticles]; //create 5 particle

float minx = -1; //Set particle range by window size

float maxx = 19;

float miny = -1;

float maxy = 19;

float minz = -1;

float maxz = 19;

float mass, x, y, z;

srand((int)time(NULL));

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

mass = ((float)((rand() % 100) + 1)) / 100.0f + 0.01f;

//x = ((float)((rand() % 100) + 1)) / 20.0f;

//y = ((float)((rand() % 100) + 1)) /100.0f + 18.0f;

z = ((float)((rand() % 100) + 1)) / 20.0f;; //set z position

P[i].Initialize( mass, Vector3(posX, posY, z)); //Initialize particle

P[i].SetBoundary( minx, maxx, miny, maxy, minz, maxz); //and set boundary of particle range

}

}

ParticleSystem::~ParticleSystem() {

delete(P);

}

void ParticleSystem::Update(float deltaTime) {

// Compute forces

Vector3 gravity(0,-9.8/10.0,0);

Vector3 wind(0.01, 0, 0);

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

Vector3 force;

force.x =gravity.x \* P[i].Mass; // f=mg

force.y =gravity.y \* P[i].Mass; // f=mg

force.z =gravity.z \* P[i].Mass; // f=mg

P[i].ApplyFriction();

P[i].ApplyForce(force);

P[i].ApplyForce(wind);

P[i].Update(deltaTime);

P[i].checkEdge();

}

}

void ParticleSystem::Draw() {

// Compute forces

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

P[i].Draw();

}

}

**<Cparticle.h>**

#include "math.h"

//It looks like interface

class Vector3 {

public :

float x, y, z;

public :

Vector3() { }

Vector3(float xx, float yy, float zz) { x = xx; y = yy; z = zz; }

void Normalization() {

float mag = sqrt(x\*x + y\*y + z\*z);

if (mag > 0.0) {

x /= mag; y /= mag; z /= mag;

}

};

void Melt(float m) { x \*= m; y \*= m; z \*= m;}

void Add(Vector3 f) { x += f.x; y += f.y; z += f.z; }

void Zero() { x = 0.0; y = 0.0; z=0.0; }

};

class Particle {

Vector3 Position; // Evolves frame to frame

Vector3 Velocity; // Evolves frame to frame

Vector3 Acceleration;

Vector3 Force; // Reset and re-computed each frame

Vector3 color ; // Particle color

float size ; // Particle size

float globalTime; //Global time

float time; // Relative time or age

float minx, miny, maxx, maxy, minz, maxz ; // boundary

public:

float Mass; // Constant

public:

Particle() { }

void Initialize(float mass, Vector3 location);

void SetBoundary(float minxx, float maxxx, float minyy, float maxyy, float minzz, float maxzz);

void Update(float deltaTime);

void Draw();

void ApplyFriction();

void ApplyForce(Vector3 &f) ;

void checkEdge();

};

**<Cparticle.cpp>**

#include <stdlib.h>

#include <GL/glut.h>

#include <GL/glu.h>

#include "CParticle.h"

void Particle::Initialize(float mass, Vector3 location) {

Mass = mass; //mass of particle

Position = location ;

//Using set velocity by random function

float vx= (((float)((rand() % 100) + 1)) / 100.0f) - 0.05f;

float vy = (((float)((rand() % 100) + 50)) / 50.0f);

float vz = (((float)((rand() % 100) + 1)) / 100.0f) - 0.05f;

Velocity = Vector3(vx, vy, vz);

Acceleration = Vector3( 0, 0, 0);

Force = Vector3( 0, 0, 0);

color = Vector3(1, 1, 1); // Particle color

size = Mass \* 2.0 ; // Particle size

}

void Particle::SetBoundary(float minxx, float maxxx, float minyy, float maxyy, float minzz, float maxzz)

{

minx = minxx;

maxx = maxxx;

miny = minyy;

maxy = maxyy;

minz = minzz;

maxz = maxzz;

}

void Particle::ApplyFriction()

{

float frictionMag = 0.01; //마¬¢Ò찰u계Æe수ùo를¬| 이I용¯e하I여¯¨Ï 마¬¢Ò찰u력¤A 조¢Ò정¢´

Vector3 friction(Velocity.x, Velocity.y, Velocity.z);

friction.Normalization(); //현o재c Velocity를¬| normalization한N다¥U. (x,y,z) = x/ Root(x2,y2,z2), Vector의C 크¨Ï기¾a는¥A 1.

friction.Melt(-1.0); //만¬¢¬약úa 정¢´해¨ª놓øo은¨¬ bound에¯¢® 넘øN어úi서ù¡©면¬e 반öY대¥e 방ö©¡향a으¢¬로¤I 가Æ¢®게ÆO한N다¥U.

friction.Melt(frictionMag);

ApplyForce(friction); //이I렇¤¢¬게ÆO 구¾¢¬한N 마¬¢Ò찰u력¤A을¡í force에¯¢® 더¥o하I는¥A 것ÆI

}

void Particle::ApplyForce(Vector3 &f)

{

Force.Add(f); //add force to preview particle force

}

void Particle::Update(float deltaTime) {

// Compute acceleration (Newton’®?s second law)

Vector3 Accel;

Accel.x =(1.0/Mass) \* Force.x;

Accel.y =(1.0/Mass) \* Force.y;

Accel.z =(1.0/Mass) \* Force.z;

// Compute new position & velocity

Velocity.x += Accel.x \* deltaTime;

Velocity.y += Accel.y \* deltaTime;

Velocity.z += Accel.z \* deltaTime;

Position.x += Velocity.x \* deltaTime;

Position.y += Velocity.y \* deltaTime;

Position.z += Velocity.z \* deltaTime;

// Zero out Force vector

Force.Zero();

}

void Particle::checkEdge() {

//만¬¢¬약úa 파¡A티¡¨ù클¡þ이I 땅Ò¡Í바öU닥¥U에¯¢® 닿¥e으¢¬면¬e 방ö©¡향a을¡í 바öU꿔÷a서ù¡© 위¡×로¤I 올¯A라Òo오¯A도ì¥ì록¤I 한N다¥U.

if (Position.x > maxx ) {

Position.x = maxx;

Velocity.x \*= -1;

} else if (Position.x < minx) {

Position.x = minx;

Velocity.x \*= -1;

}

if (Position.y > maxy ) {

Position.y = maxy;

Velocity.y \*= -1;

} else if (Position.y < miny) {

Position.y = miny;

Velocity.y \*= -1;

}

if (Position.z > maxz ) {

Position.z = maxz;

Velocity.z \*= -1;

} else if (Position.z < minz) {

Position.z = minz;

Velocity.z \*= -1;

}

}

void Particle::Draw() {

//Rendering a Particle

glColor4f(1.0f, 1.0f, 1.0f, 0.5f);

glPushMatrix();

glTranslatef( Position.x, Position.y, Position.z );

glutSolidSphere(Mass, 20, 20);

glPopMatrix();

}