НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «Київський Політехнічний Інститут»

Факультет Інформатики і Обчислювальної Техніки

Кафедра Обчислювальної Техніки

*Системи реального часу*

***Лабораторна pобота №1***

*Генерація випадкових карт*

Виконали: студенти 3 курсу

гр. ІП-12,

Скрипник Андрій,

Табунченко Микола,

Герасимчук Олексій,

Гринчук Андрій,

Гоц Олексій

Київ 2013

**Завдання**

Створити програмний модуль для генерації випадкових карт місцевості. У вигляді 3D прямокутників представити будинки, які і будуть виступати на даному етапі основними перешкодами для літаків.

**Теоретичні відомості**

Клас ***Aircraft*** представляє віртуальну модель літака. Модель матиме висоту, ширину, довжину та швидкість. Клас має конструктор та метод getID, який повертає ID літака.

Клас ***Vector3D*** призначений для визначення координат вектора напрямку руху літака. Оскільки у нас 3D простір, вектор матиме x, y та z координати. Клас має констуктор, а також була використана інкапсуляція для безпеки даних.

Клас ***Simulator*** призначений для генерації віртуального середовища, тобто карти та літаків. Ми маємо можливість задати кількість літаків, будинків, а також розмір карти у метрах. У тілі класу відбувається генерація випадкових 3D об’єктів. Об’єкти не повинні виходити за межі карти.

Клас ***Map*** відповідає безпосередньо за створення самої карти. Карта матиме лише дві координати x, y. Висота об’єкта зазначатиметься в комірках матриці карти. Клас має 2 конструктора, а також метод для додавання об’єктів, який повертає булеановське значення про результат додавання об’єкта.

Клас ***Graphics*** відповідає за візуальне представлення об’єктів на карті

Він є абстрактним базовим класом для всіх контекстів графіки, які дозволяють додатку малювати компоненти, які реалізуються на різних пристроях, а також на закадрових зображеннях.

Графічний об'єкт інкапсулює стан інформації, необхідної для основних операцій рендеринга, що підтримує Java. Ця інформація про стан включає в себе наступні характеристики:

* Компонент-об'єкт, на якому малювати.
* Переклад походження для надання і відсікання координат.
* Поточний кліп.
* Поточний колір.
* Поточний шрифт.
* Логічна функція пікселів (XOR або фарби).

**Код програми**

***class Aircraft***

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package simulator;

/\*\*

\*

\* @author AnarH

\*/

public class Aircraft {

private int id;

private int width = 30;

private int height = 10;

private int length = 40;

private int speed = 100;

public Vector3D vector3d;

public Aircraft(int id, Vector3D vector3d){

this.id = id;

this.vector3d = vector3d;

}

public int getID(){

return this.id;

}

}

***class Vector3D***

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package simulator;

/\*\*

\*

\* @author AnarH

\*/

public class Vector3D {

int x1;

int y1;

int z1;

int x2;

int y2;

int z2;

public Vector3D(int x1, int y1, int z1, int x2, int y2, int z2) {

this.x1 = x1;

this.y1 = y1;

this.z1 = z1;

this.x2 = x2;

this.y2 = y2;

this.z2 = z2;

}

public int getX1() {

return x1;

}

public void setX1(int x1) {

this.x1 = x1;

}

public int getY1() {

return y1;

}

public void setY1(int y1) {

this.y1 = y1;

}

public int getZ1() {

return z1;

}

public void setZ1(int z1) {

this.z1 = z1;

}

public int getX2() {

return x2;

}

public void setX2(int x2) {

this.x2 = x2;

}

public int getY2() {

return y2;

}

public void setY2(int y2) {

this.y2 = y2;

}

public int getZ2() {

return z2;

}

public void setZ2(int z2) {

this.z2 = z2;

}

}

***class Simulator***

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package simulator;

import com.sun.j3d.utils.geometry.ColorCube;

import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.BranchGroup;

/\*\*

\*

\* @author AnarH

\*/

public class Simulator {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

int mapsize = 2000;

int aircraftAmmount = 50;

int objectAmmount = 100;

//int [][]m = map.getMap();

float objectMap[][] = new float[objectAmmount][5];

Map map = new Map(mapsize);

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

float x = (int)(Math.random() \* mapsize-1)+1;

float y = (int)(Math.random() \* mapsize-1)+1;

float width = (int)(Math.random() \* 100)+1;

float height = (int)(Math.random() \* 100)+1;

float length = (int)(Math.random() \* 200)+1;

map.addObject((int)x, (int)y, (int)width, (int)height, (int)length);

objectMap[i][0] = x / 100;

objectMap[i][1] = y / 100;

objectMap[i][2] = width / 100;

objectMap[i][3] = height / 100;

objectMap[i][4] = length / 100;

}

Graphics graphics = new Graphics(objectMap);

}

}

***class Map***

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package simulator;

/\*\*

\*

\* @author AnarH

\*/

public class Map {

private int[][] map;

public Map(int mapSizeX) {

this.map = new int[mapSizeX][mapSizeX];

}

public Map(int mapSizeX, int mapSizeY) {

this.map = new int[mapSizeX][mapSizeY];

}

public boolean addObject(int x, int y, int width, int length, int height) {

if ((x - width) > 0 || (x + width) < map.length || (y - length) > 0 || (y + length) < map[0].length) {

for (int i = x - (width / 2); i < x + (width / 2); i++) {

for (int j = y - (length / 2); j < y + (length / 2); j++) {

try{

this.map[i][j] = height;

}catch(Exception e){

System.err.println("Object is outside this map.");

}

}

}

return true;

}

return false;

}

public int[][] getMap() {

return this.map;

}

}

***class Graphics***

package simulator;

import java.awt.Color;

import com.sun.j3d.utils.geometry.GeometryInfo;

import com.sun.j3d.utils.geometry.NormalGenerator;

import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.\*;

import javax.vecmath.\*;

import com.sun.j3d.utils.geometry.ColorCube;

import com.sun.j3d.utils.geometry.Cone;

import com.sun.j3d.utils.geometry.Cylinder;

import com.sun.j3d.utils.geometry.Sphere;

// An Egyptian pyramid

// Base divided into two triangles

public class Graphics {

public Graphics(float[][] objectMap) {

SimpleUniverse universe = new SimpleUniverse();

BranchGroup group = new BranchGroup();

float size = 30;

// X axis made of spheres

for (float x = .0f; x <= size; x = x + 1f) {

Sphere sphere = new Sphere(0.05f);

TransformGroup tg = new TransformGroup();

Transform3D transform = new Transform3D();

Vector3f vector = new Vector3f(x, .0f, .0f);

transform.setTranslation(vector);

tg.setTransform(transform);

tg.addChild(sphere);

group.addChild(tg);

}

// Y axis made of cones

for (float y = .0f; y <= size; y = y + 1f) {

TransformGroup tg = new TransformGroup();

Transform3D transform = new Transform3D();

Cone cone = new Cone(0.05f, 0.1f);

Vector3f vector = new Vector3f(.0f, y, .0f);

transform.setTranslation(vector);

tg.setTransform(transform);

tg.addChild(cone);

group.addChild(tg);

}

// Z axis made of cylinders

for (float z = .0f; z <= size; z = z + 1f) {

TransformGroup tg = new TransformGroup();

Transform3D transform = new Transform3D();

Cylinder cylinder = new Cylinder(0.05f, 0.1f);

Vector3f vector = new Vector3f(.0f, .0f, z);

transform.setTranslation(vector);

tg.setTransform(transform);

tg.addChild(cylinder);

group.addChild(tg);

}

for (int i = 0; i < objectMap.length; i++) {

float posX = objectMap[i][0];

float posZ = objectMap[i][1];

float width = objectMap[i][2];

float length = objectMap[i][3];

float height = objectMap[i][4];

for (float x = posX; x <= posX + width; x += .1f) {

for (float y = 0; y <= height; y += .1f) {

for (float z = posZ; z <= posZ + length; z += .1f) {

TransformGroup tg = new TransformGroup();

Transform3D transform = new Transform3D();

ColorCube cube = new ColorCube(0.05);

Vector3f vector = new Vector3f(x, y, z);

transform.setTranslation(vector);

tg.setTransform(transform);

tg.addChild(cube);

group.addChild(tg);

}

}

}

}

Color3f light1Color = new Color3f(.1f, 1.4f, .1f); // green light

BoundingSphere bounds = new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0);

Vector3f light1Direction = new Vector3f(40.0f, -70.0f, -120.0f);

DirectionalLight light1 = new DirectionalLight(light1Color, light1Direction);

light1.setInfluencingBounds(bounds);

group.addChild(light1);

universe.getViewingPlatform().setNominalViewingTransform();

// add the group of objects to the Universe

universe.addBranchGraph(group);

// above pyramid

Vector3f viewTranslation = new Vector3f();

viewTranslation.z = 36;

viewTranslation.x = 10;

viewTranslation.y = -5;

Transform3D viewTransform = new Transform3D();

viewTransform.setTranslation(viewTranslation);

Transform3D rotation = new Transform3D();

rotation.rotX(-0.5);

//rotation.rotY(-1.5);

rotation.mul(viewTransform);

//group.addChild(new ColorCube(0.3));

universe.getViewingPlatform().getViewPlatformTransform().setTransform(rotation);

universe.getViewingPlatform().getViewPlatformTransform().getTransform(viewTransform);

// lights

light1.setInfluencingBounds(bounds);

group.addChild(light1);

Color3f ambientColor = new Color3f(.4f, .4f, .4f);

AmbientLight ambientLightNode = new AmbientLight(ambientColor);

ambientLightNode.setInfluencingBounds(bounds);

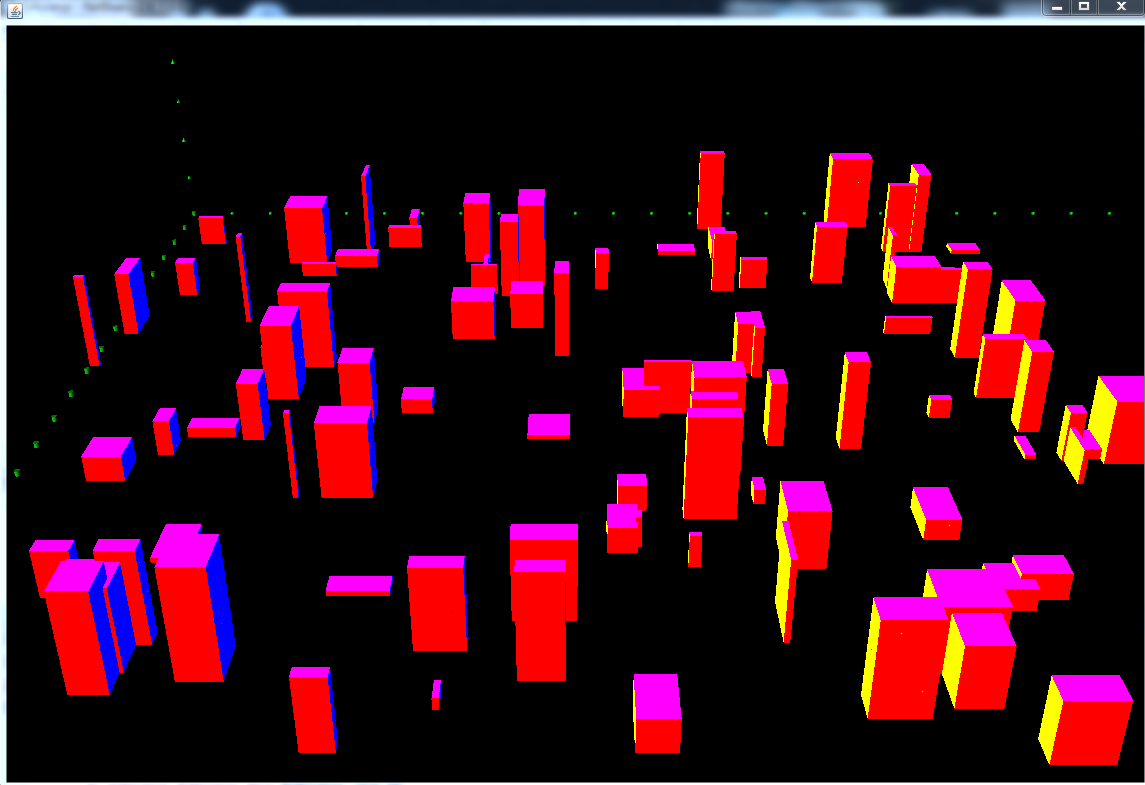
group.addChild(ambientLightNode);

universe.addBranchGraph(group);

}

}

**Скріншот**



**Висновки**

Дана лабораторна робота є хорошим стартом для нашого проекту. Нам вдалося створити карту з випадковою генерацією 3D об’єктів. 3D прямокутники виступатимуть у ролі уявних будинків. На даному етапі будинки - це єдині перешкоди для віртуальних літаків. Літаки уникатимуть зіткнення з ними лише вправо або вліво. В наступних лабораторних буде реалізована можливість огинання перешкоди по вертикалі.