**Поясненные содержания о программе**

**Установленная и используемая среда**: OpenGL

Установка OpenGL в среду Visual Studio:

Первый - загрузить OpenGL по ссылке: http://www.mediafire.com/file/4mxl8e14pa6e8il/glut-3.7.6-bin.rar/file

Тогда вам просто нужно найти и скопировать скачанные файлы и запомнить:

+ Заголовок скопирован для включения

+ Lib copypy в папку lib

и файл dll, который мы все еще копируем в: C: \ Windows \ System32

Затем перейдите в Visual Studio, чтобы создать проект, и наведите указатель мыши на проект, выберите свойства, выберите компоновщик, выберите ввод и добавьте файл библиотеки следующим образом:

glut32.lib, glu32.lib, opengl32.lib

Затем нам просто нужно объявить #include <glut.h> в объявлении библиотеки для нормальной работы.

**Пояснение к коду:**

-Включает два файла (1 основной файл кода и 1 вспомогательный файл для описания эффекта дождя (Rain\_Effect.h)

-Часть файла растрового изображения может быть удалена без каких-либо изменений (используется для загрузки изображений в текстуру, но мы можем использовать функцию LoadGLTexture ниже)

**Основной код построен по модели:**

-Рисовать электрические столбы, солнечные батареи (функция drawShohidMinar)

-Нарисовать вентилятор самолета (вентилятор), если хотите, чтобы 2 вентилятора использовали челюсти (fan2) - Изменение цвета самолета непрерывно автоматически (функция таймера) -Нарисуйте самолет (функция самолета)

-Нарисуйте одиночный дом (функция singleTolaHouse) -Сплошной чертеж дома (функция дома)

-Нарисовать землю для размещения столбов электричества, солнечных батарей (функция soheedMinarEnv)

-Нарисовать общую среду (функция среды) -Draw вызывает все объекты в (функция рисования)

-Функция рисования длинного текста (drawStrokeText, drawStrokeText2)

-Функция рисования персонажа (drawStrokeChar)

-Функция, которая представляет все вышеперечисленное (функция отображения)

- Функция для обработки операций с клавиатурой (функция клавиш)

**Подробное объяснение кода:**

**Объявление библиотеку для использования, объявите начальное значение для переменной EN\_SIZE (значение переменной ниже).**

**Справочные файлы:** <https://github.com/rksazid/PlaneGame-OpenGL-CPP>

**Идея**: рисовать объекты в коде с помощью встроенных функций, таких как: glutsolidCube, glutsolidSphere, glutSolidTorus, ....., а затем с помощью glScalef, glRotatef для получения желаемого объекта и эффекта.

**Код после редактирования, добавления эффектов, обновления:**

#include<windows.h>

#include<glut.h>

#include<gl/GLAUX.H>

#pragma comment(lib, "glaux.lib")

//#include<bits/stdc++.h>

#include<iostream>

#include<vector>

#include<string>

#include <stdlib.h>

#include<Windows.h>

#include<mmsystem.h>

#define EN\_SIZE 20

#include "RGBpixmap.h"

#include "Rain\_Effect.h"

using namespace std;

// Использование растрового изображения как способ присвоения объектам 2D, 3D изображений (в настоящее время я буду использовать glTexture

RGBpixmap pix[6];

/ \* Объясняем значение используемых параметров:

-color1, color2, color3: параметр для настройки изменения цвета

-r [], g [], b []: массив, содержащий 10 значений настройки цвета для дома

-EN\_SIZE: Задает размер для определения земли (при увеличении анимация замедляется)

-torusPosx, torusPosy - это массивы, содержащие координаты x, y круговых перекрестков в окружающей среде.

-tola: двумерный массив, содержащий соответствующие установочные значения для параметра жилищного строительства (function house, singletolaHouse)

-TIME: значение присваивается t, причем t изначально назначается каждый раз при повторной инициализации

-rotX, rotY, rotZ: указанные значения вращаются в направлениях x, y, z плоскости

-speed: установка значения скорости самолета.

-tZ до tZ6: значение настройки для переменных, работающих со скоростью, соответствующей каждому изменению среды с tZ

-angleBackFrac: соответствующий угол изменения для rotX, rotY, rotZ плоскости

-START: значение установки запускает программу

-rot: начальная установка ложного значения для управления поворотом экрана самолета.

-zoom: установка значения каждый раз, когда я приближаюсь к объекту

-tX, tY: расположение плоскости

-angle: значение настройки изменяет угол наклона плоскости, когда я нажимаю кнопки перемещения

\*/

float zoom = 4;

int tola[5000][5000];

float tX = 0, tY = 0, tZ = -8, rX = 0, rY = 0, rZ = 4;

float tZ1 = -20, tZ2 = -40, tZ3 = -60, tZ4 = -80, tZ5 = -100, tZ6 = -120;

float rotX = 0, rotY = 0, rotZ = 0;

//float cosX = 0, cosY = 1, cosZ = 0;

float angle = 0;

//float xEye = 0.0f, yEye = 5.0f, zEye = 30.0f;

//float cenX = 0, cenY = 0, cenZ = 0, roll = 0;

//float radius = 0;

//float theta = 0, slope = 0;

float speed = 0.3;

float angleBackFrac = 0.2;

bool saheedMinarVisible = false;

float r[] = { 0.1,0.4,0.0,0.9,0.2,0.5,0.0,0.7,0.5,0.0 };

float g[] = { 0.2,0.0,0.4,0.5,0.2,0.0,0.3,0.9,0.0,0.2 };

float b[] = { 0.4,0.5,0.0,0.7,0.9,0.0,0.1,0.2,0.5,0.0 };

int TIME = 0;

bool START = false;

float torusPosX[7] = { 1,-2,3,-4,-2,0,2 };

float torusPosY[7] = { 2,3,10,6,7,4,1 };

float color1 = 0.0f;

float color2 = 0.0f;

float color3 = 0.0f;

bool rot = false;

GLuint texture; // управление текстурой

char texture\_name[100]={"brics.bmp"}; // сохранить имя файла текстуры

// Функция загрузки текстуры

bool LoadGLTextures()

{

int ret = false;

AUX\_RGBImageRec \*texture\_image = NULL;

if (texture\_image = LoadBMP(texture\_name))

{

glGenTextures(1, &texture); // Запустить процесс создания текстуры.

glBindTexture(GL\_TEXTURE\_2D, texture);

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

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

// сопоставить данные растрового изображения с текстурой.

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, texture\_image->sizeX,

texture\_image->sizeY, 0, GL\_RGB,

GL\_UNSIGNED\_BYTE, texture\_image->data);

}

else

{

ret = false;

if (texture\_image)

{

if (texture\_image->data)

free(texture\_image->data);

free(texture\_image);

}

}

return ret;

}

// Функция для установки значения рамки проекции, вида объекта

static void resize(int width, int height)

{

const float ar = (float)width / (float)height;

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glFrustum(-ar, ar, -1.0, 1.0, 2.0, 1000.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

// Нарисовать набор солнечных столбов

void drawShohidMinar() {

// Выполнить розыгрыш подиума (по очереди от нижнего подиума к верхнему)

glColor3d(0.4, 0.2, 0.2);//sử dụng màu nâu

glPushMatrix();

glTranslated(0, 1.55, 0);

glScaled(2, 0.05, 1.5);

glutSolidCube(1);

glPopMatrix();

glColor3d(0.4, 0.2, 0.2);

glPushMatrix();

glTranslated(0, 1.6, 0);

glScaled(1.9, 0.05, 1.4);

glutSolidCube(1);

glPopMatrix();

//glColor3d(0.4,0.2,0.2);

//glColor3d(1,0.8,0.7);

glColor3d(0.4, 0.2, 0.2);

glPushMatrix();

glTranslated(0, 1.65, 0);

glScaled(1.8, 0.05, 1.3);

glutSolidCube(1);

glPopMatrix();

/// пьедестал панели солнечных батарей

glColor3d(1, 1, 1);

glPushMatrix();

glTranslated(0, 1.68, -0.4);

glScaled(0.5, 0.02, 0.08);

glutSolidCube(1);

glPopMatrix();

/// Центральный полюс (главной солнечной панели)

glPushMatrix();

glTranslated(0, 1.99, -0.4);

glScaled(0.06, 0.7, 0.04);

glutSolidCube(1);

glPopMatrix();

/// Постоянный провод этой опоры (порядок рисования: 3 провода сверху, 3 провода снизу)

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(0.07, 1.99, -0.4);

glScaled(0.003, 0.7, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.11, 1.99, -0.4);

glScaled(0.003, 0.7, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.15, 1.99, -0.4);

glScaled(0.003, 0.7, 0.003);

glutSolidCube(1);

glPopMatrix();

///

glPushMatrix();

glTranslated(-0.22, 0, 0);

glPushMatrix();

glTranslated(0.07, 1.99, -0.4);

glScaled(0.003, 0.7, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.11, 1.99, -0.4);

glScaled(0.003, 0.7, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.15, 1.99, -0.4);

glScaled(0.003, 0.7, 0.003);

glutSolidCube(1);

glPopMatrix();

glPopMatrix();

/// Горизонтальный провод этого столба

glPushMatrix();

glTranslated(2.2, 0, -0.1); // репозиция относительно стоячей проволоки

glScaled(4.2, 1, 1);

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.528, 1.85, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2.02, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2.18, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glColor3d(1, 1, 1);

glPopMatrix();

/// Самый дальний провод этого полюса

glColor3d(1, 1, 1);

glPushMatrix();

glTranslated(-0.22, 1.99, -0.4);

glScaled(0.06, 0.7, 0.04);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.22, 1.99, -0.4);

glScaled(0.06, 0.7, 0.04);

glutSolidCube(1);

glPopMatrix();

/// Построить столбы по диагонали поверх солнечной панели. glPushMatrix();

glTranslated(0, 0.743, -1.424); //thiết lập lại vị trí so với những cột điện đứng

glRotated(45, 1, 0, 0);

//glColor3d(1,0,1);

glPushMatrix();

glTranslated(0, 1.99, -0.4);

glScaled(0.06, 0.3, 0.04);

glutSolidCube(1);

glPopMatrix();

//glColor3d(1,1,1);

glPushMatrix();

glTranslated(-0.22, 1.99, -0.4);

glScaled(0.06, 0.3, 0.04);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.22, 1.99, -0.4);

glScaled(0.06, 0.3, 0.04);

glutSolidCube(1);

glPopMatrix();

// Устройство горизонтального ограждения верхнего полюса мощности

glPushMatrix();

glTranslated(0, 2.15, -0.4);

glScaled(0.5, 0.04, 0.04);

glutSolidCube(1);

glPopMatrix();

/// Сделать вертикальные провода для столбов выше

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(0.07, 1.99, -0.4);

glScaled(0.003, 0.277, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.11, 1.99, -0.4);

glScaled(0.003, 0.277, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.15, 1.99, -0.4);

glScaled(0.003, 0.277, 0.003);

glutSolidCube(1);

glPopMatrix();

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.22, 0, 0);//thiết lập lại vị trí của những dây điện đứng ở phía bên kia

glPushMatrix();

glTranslated(0.07, 1.99, -0.4);

glScaled(0.003, 0.277, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.11, 1.99, -0.4);

glScaled(0.003, 0.277, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.15, 1.99, -0.4);

glScaled(0.003, 0.277, 0.003);

glutSolidCube(1);

glPopMatrix();

glPopMatrix();

/// Сделать горизонтальные провода для верхних панелей (включая 2 горизонтальных провода в центре).

glPushMatrix();

glTranslated(2.2, 0, -0.1);

glScaled(4.2, 1, 1);

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.528, 1.85, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

/\*glPushMatrix();

glTranslated(-0.528, 2.15, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();\*/

glColor3d(1, 1, 1);

glPopMatrix(); //câu lệnh này liên kết với glPushMatrix() ở phía trên trong phần khung phía trên

glPopMatrix();

/\* НАЧАТЬ СТРОИТЕЛЬСТВО ЭЛЕКТРИЧЕСКИХ опор возле главной опоры \*/

/// Построить первый левый столб (в порядке построения блока переднего края к нижнему горизонтальному блоку).

glColor3d(1, 1, 1);

glPushMatrix();

glTranslated(0.1, 0, -0.4); //Bắt đầu thực hiện đặt cột điện nghiêng góc 45 độ so với cột điện chính

glRotated(45, 0, 1, 0);

glPushMatrix();

glTranslated(-0.605, 1.94, -0.3);

glScaled(0.045, 0.65, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.45, 1.94, -0.3);

glScaled(0.045, 0.65, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2.258, -0.3);

glScaled(0.199, 0.04, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 1.68, -0.3);

glScaled(0.199, 0.02, 0.06);

glutSolidCube(1);

glPopMatrix();

/// Сделать постоянный провод для этого столба

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.64, -0.05, 0.1);

glScaled(1, 1.02, 1);

glPushMatrix();

glTranslated(0.078, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.11, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.145, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPopMatrix();

/// Горизонтальный провод этого столба

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.528, 1.85, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2.15, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glColor3d(1, 1, 1);

glPopMatrix();

/// Построить второй левый полюс

glPushMatrix();

glTranslated(0.65, 0, 0.3);

glRotated(-45, 0, 1, 0);

glPushMatrix();

glTranslated(-0.605, 1.94, -0.3);

glScaled(0.045, 0.65, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.45, 1.94, -0.3);

glScaled(0.045, 0.65, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2.258, -0.3);

glScaled(0.199, 0.04, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 1.68, -0.3);

glScaled(0.199, 0.02, 0.06);

glutSolidCube(1);

glPopMatrix();

/// Сделать постоянный провод для этого столба

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.64, -0.05, 0.1);

glScaled(1, 1.02, 1);

glPushMatrix();

glTranslated(0.078, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.11, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.145, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPopMatrix();

glColor3d(1, 1, 1);

/// Сделать для этого столба горизонтальные провода.

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.528, 1.85, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2.15, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glColor3d(1, 1, 1);

glPopMatrix();

/// Построить столб мощности, расположенный в первом правом углу.

glPushMatrix();

/// pasher piller left 1

glTranslated(0.06, 0, 0.14);

glPushMatrix();

glTranslated(-0.2, 0, -0.31);

glRotated(45, 0, 1, 0);

glPushMatrix();

glTranslated(-0.605, 1.88, -0.3);

glScaled(0.045, 0.4, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.45, 1.88, -0.3);

glScaled(0.045, 0.4, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2.08, -0.3);

glScaled(0.2, 0.04, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 1.68, -0.3);

glScaled(0.199, 0.02, 0.06);

glutSolidCube(1);

glPopMatrix();

/// Сделать постоянный провод для этого столба

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.641, 0.43, 0.1);

glScaled(1, 0.73, 1);

glPushMatrix();

glTranslated(0.078, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.11, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.145, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPopMatrix();

/// Сделать для этого столба горизонтальные провода.

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.528, 1.8, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 1.96, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glColor3d(1, 1, 1);

///ROD

glPopMatrix();

/// Построить второй правый столб

glPushMatrix();

glTranslated(0.83, 0, 0.39);

glRotated(-45, 0, 1, 0);

glPushMatrix();

glTranslated(-0.605, 1.88, -0.3);

glScaled(0.045, 0.4, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.45, 1.88, -0.3);

glScaled(0.045, 0.4, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 2.1, -0.3);

glScaled(0.199, 0.04, 0.03);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 1.68, -0.3);

glScaled(0.199, 0.02, 0.06);

glutSolidCube(1);

glPopMatrix();

/// Сделать для этого столба горизонтальные провода.

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.528, 1.8, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-0.528, 1.96, -0.3);

glScaled(0.1, 0.003, 0.003);

glutSolidCube(1);

glPopMatrix();

glColor3d(1, 1, 1);

/// Сделать постоянный провод для этого столба

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.641, 0.43, 0.1);

glScaled(1, 0.73, 1);

glPushMatrix();

glTranslated(0.078, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.11, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(0.145, 1.99, -0.4);

glScaled(0.003, 0.56, 0.003);

glutSolidCube(1);

glPopMatrix();

glPopMatrix();

glColor3d(1, 1, 1);

glPopMatrix();

glPopMatrix();

/// Круг

glColor3d(1, 0, 0);

glPushMatrix();

glTranslated(0, 2.1, -0.44);

glScaled(0.35, 0.35, 0.01);

glutSolidSphere(1, 50, 50);

glPopMatrix();

//glColor3d(0, 0, 0);

//glPushMatrix();

//glTranslated(-0.18, 1.9, -0.45);

//glScaled(0.01, 0.5, 0.01);

//glutSolidCube(1);

//glPopMatrix();

//glColor3d(0, 0, 0);

//glPushMatrix();

//glTranslated(0.18, 1.9, -0.45);

//glScaled(0.01, 0.5, 0.01);

//glutSolidCube(1);

//glPopMatrix();

}

// Функция рисования веера самолета: если в 2 блоках используется только первый блок -> 1 крыло; используйте блок 2-> 2 крылаvoid fan() {

//glColor3d(0.5, 1, 0);

//glPushMatrix();

//glTranslated(0, 0, 0);

//glScaled(1, 1, 0.7);

//glutSolidSphere(0, 30, 30);

//glPopMatrix();

glColor3d(0.5, 1, 0); //màu vàng lục

glPushMatrix();

glTranslated(2.5, 0, 0);

//glRotated(5, 0, 1, 0);

glScaled(0.2, 1.5, 0.05);

glutSolidSphere(1, 30, 30);

glPopMatrix();

}

void fan2()

{

glColor3d(0.5, 1, 0);

glPushMatrix();

//glTranslated(0, 0, 0);

glTranslated(2.5, 0, 0);

//glRotated(-5, 0, 1, 0);

//glRotated(90, 0, 0, 1);

glScaled(0.2, 1.5, 0.05);

glutSolidSphere(1, 30, 30);

glPopMatrix();

}

// Функция изменения цвета фюзеляжа

**Идея:** изменения цвета заключается в том, что мы увеличиваем значение цвета плоскости каждый раз, когда изменяется значение угла плоскости.

void timer(int value)

{

angle += 3.0f;

//changeColor?

color1 += 0.001f;

color2 += 0.003f;

color3 += color2;

if (color1 > 1.0)

color1 = 0;

if (color2 > 1.0)

color2 = 0;

if (color3 > 1.0)

color3 = 0;

glutPostRedisplay();

glutTimerFunc(16, timer, 0);

}

//Hàm vẽ người

//Hàm vẽ mưa

/\*Variables used in this function\*/

/\*

t: значение для получения инициализаций при запуске программы

a: расчет угла вращения вентилятора

\*/

// Функция реализации для рисования плоскости

void plane() {

const double t = glutGet(GLUT\_ELAPSED\_TIME) / 1000.0;

double a = t \* 90.0;

/// Основной корпус самолета:

//glColor3d(0.5, 1, 0);

glColor3f(color1, color2, color3); //gọi tới giá trị thay đổi màu của thân máy bay

glPushMatrix();

glTranslated(0, 0, 0);

glScaled(3, 0.4, 0.5);

glutSolidSphere(1, 30, 30);

glPopMatrix();

// Часть кабины:

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(1.7, 0.1, 0);

glScaled(1.5, 0.7, 0.8);

//glRotated(40, 0, 1, 0);

glutSolidSphere(0.45, 30, 30);

glPopMatrix();

/// Правая часть крыла: здесь используется glRotatef намеренно, когда самолет вращается.

glColor3d(0.8, 1, 0);

glPushMatrix();

glTranslated(0, 0, 1.2);

glRotated(-50, 0, 1, 0);

glScaled(0.7, 0.1, 3);

glRotated(25, 0, 1, 0);

glutSolidCube(1);

glPopMatrix();

// Деталь двигателя под правым крылом

// Подкрыльевая часть двигателя (внутри)

glColor3d(0.8, 1, 0);

glPushMatrix();

glTranslated(-0.3, -0.15, 1.5);

glRotated(90, 0, 1, 0);

glScaled(0.1, 0.1, 0.9);

glutSolidTorus(0.5, 0.5, 50, 50);

glPopMatrix();

// Секция двигателя под крылом (снаружи)

glColor3d(0.8, 1, 0);

glPushMatrix();

glTranslated(0.2, -0.15, 0.9);

glRotated(90, 0, 1, 0);

glScaled(0.1, 0.1, 0.9);

glutSolidTorus(0.5, 0.5, 50, 50);

glPopMatrix();

/// вентилятор самолета

// glPushMatrix();

// glTranslated(0,0,0.5);

// glRotated(10\*a,0,0,1);

// glScaled(0.1,0.1,0.1);

// fan();

// glPopMatrix();

/// Левая часть крыла

glColor3d(0.8, 1, 0);

glPushMatrix();

glTranslated(0, 0, -1.2);

glRotated(50, 0, 1, 0);

glScaled(0.7, 0.1, 3);

glRotated(-25, 0, 1, 0);

glutSolidCube(1);

glPopMatrix();

// Деталь двигателя под левым крылом

glColor3d(0.8, 1, 0);

glPushMatrix();

glTranslated(-0.3, -0.15, -1.5);

glRotated(90, 0, 1, 0);

glScaled(0.1, 0.1, 0.9);

glutSolidTorus(0.5, 0.5, 50, 50);

glPopMatrix();

glColor3d(0.8, 1, 0);

glPushMatrix();

glTranslated(0.2, -0.15, -0.9);

glRotated(90, 0, 1, 0);

glScaled(0.1, 0.1, 0.9);

glutSolidTorus(0.5, 0.5, 50, 50);

glPopMatrix();

/// Рисовать хвост самолета

glPushMatrix();

glTranslated(-2.8, 0, 0);

glScaled(0.8, 0.5, 0.3);

/// Хвостовой самолет правый

glColor3d(0.8, 1, 0);

glPushMatrix();

glTranslated(0.4, 0, 1.5);

glRotated(-30, 0, 1, 0);

glScaled(0.7, 0.1, 3);

glRotated(10, 0, 1, 0);

glutSolidCube(1);

glPopMatrix();

/// Левый хвост самолета

glColor3d(0.8, 1, 0);

glPushMatrix();

glTranslated(0.4, 0, -1.5);

glRotated(30, 0, 1, 0);

glScaled(0.7, 0.1, 3);

glRotated(-10, 0, 1, 0);

glutSolidCube(1);

glPopMatrix();

glPopMatrix();

/// Верхний хвост

glColor3d(0.8, 1, 0);

glPushMatrix();

glTranslated(-2.7, 0.5, 0);

glRotated(45, 0, 0, 1);

glScaled(0.8, 2, 0.1);

glRotated(-20, 0, 0, 1);//sử dụng này để nó nghiêng đẹp hơn

glutSolidCube(0.5);

glPopMatrix();

// glColor3d(0.8,1,0);

// glPushMatrix();

// glTranslated(-2.95,0.85,0);

// glRotated(90,0,1,0);

// glScaled(0.05,0.05,0.6);

// glutSolidTorus(0.5,0.5,50,50);

// glPopMatrix();

/// Вентилятор самолета

// Если 1 вентилятор:

glPushMatrix();

glTranslated(0, 0, 0);

glRotated(10 \* a, 1, 0, 0);

//glRotated(90,1,0,0);

fan();

glPopMatrix();

// Если 2 вентилятор:

// glPushMatrix();

// glTranslated(0,0,0);

//glPushMatrix();

// glRotated(10\*a,1,0,0);

// //glRotated(90,1,0,0);

// fan();

//glPopMatrix();

//glPushMatrix();

//glRotated(-(10 \* a), 1, 0, 0);

//fan2();

//glPopMatrix();

// glPopMatrix();

}

/\* Функции для черчения домиков: singleTolaHouse and house \*/

void singleTolaHouse(int R, int G, int B) {

glColor3d(r[R % 11], g[G % 11], b[B % 11]);

// Строить блоки по очереди

// Детский блок

glPushMatrix();

glTranslated(0, 0, 0);

glutSolidCube(1);

glPopMatrix();

// Два блока средней-высокой

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(0.2, 0, 0);

glScaled(0.3, 0.3, 1.001);

glutSolidCube(1);

glPopMatrix();

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(-0.2, 0, 0);

glScaled(0.3, 0.3, 1.001);

glutSolidCube(1);

glPopMatrix();

// Два высоких блока позади

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(0, 0, 0.2);

glScaled(1.001, 0.3, 0.3);

glutSolidCube(1);

glPopMatrix();

glColor3d(0, 0, 0);

glPushMatrix();

glTranslated(0, 0, -0.2);

glScaled(1.001, 0.3, 0.3);

glutSolidCube(1);

glPopMatrix();

}

/\* Функция рисования дома и анимации непрерывного отображения дома (устанавливается в цикле через glScalef) \*/

void house(int n, int R, int G)

{

for (int i = 0; i < n; i++)

{

glPushMatrix();

glTranslated(0, 0.8+i, 0); // Расстояние между домами (в переводе ширина-y-размер)

singleTolaHouse(G, R, i); // Если я изменю порядок G, R, i, я получу разные цвета

glPopMatrix();

}

}

// Нарисовать 2 противоположные площадки, на которых расположены столбы.

void soheedMinarEnv() {

/// Vùng đất,bề mặt ở dưới

glColor3d(0, 0.5, 0.1);

glPushMatrix();

glTranslated(0, 0, 0);

glScaled(EN\_SIZE \* 2, 0.3, EN\_SIZE \* 2);

glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glTranslated(-8, -2.7, -5);

glRotated(65, 0, 1, 0);

//glRotated(15,0,1,0);

glScaled(2, 2, 2);

drawShohidMinar();

glPopMatrix();

glPushMatrix();

glTranslated(8, -2.7, -5);

glRotated(-65, 0, 1, 0);

//glRotated(15,0,1,0);

glScaled(2, 2, 2);

drawShohidMinar();

glPopMatrix();

}

// Выполнить рисование среды

void environment(int n) {

// Вызов функции рисования дождя

glPushMatrix();

drawRain();

glPopMatrix();

/// Призывая землю, поверхность ниже и здесь загрузим текстуру, чтобы создать непрерывный мигающий эффект

//glColor3d(0, 0.5, 0.1);

LoadGLTextures();

glPushMatrix();

glTranslated(0, 0, 0);

glScaled(EN\_SIZE \* 2, 0.3, EN\_SIZE \* 2);

glTexCoord2f(EN\_SIZE \* 2, 0.3);

glutSolidCube(1);

glPopMatrix();

// Сделать зеленые круги

glColor3d(0, 1, 0.1);

glPushMatrix();

glTranslated(torusPosX[n], torusPosY[n], 0);

glScaled(0.3, 0.3, 0.3);

glutSolidTorus(1, 3, 30, 30);

glPopMatrix();

// Выполнить командный блок для вызова компонентов

for (int i = -(EN\_SIZE / 2) + 1; i < (EN\_SIZE / 2); i += 2) {

for (int j = -(EN\_SIZE / 2) + 1; j < (EN\_SIZE / 2); j += 2) {

if (tola[i + (EN\_SIZE / 2) + 1][j + (EN\_SIZE / 2) + 1] != 0) {

glPushMatrix();

glTranslated(i, 0, j);

house(tola[i + (EN\_SIZE / 2) + 1][j + (EN\_SIZE / 2) + 1], i, j);

glPopMatrix();

}

else if (i >= -5 && i <= 5) {

}

else {

tola[i + (EN\_SIZE / 2) + 1][j + (EN\_SIZE / 2) + 1] = (rand() % 5) + 1;

glPushMatrix();

glTranslated(i, 0, j);

house(tola[i + (EN\_SIZE / 2) + 1][j + (EN\_SIZE / 2) + 1], i, j);

glPopMatrix();

}

}

}

// glColor3d(0,1,0.7);

// glPushMatrix();

// glRotated(angle,0,1,0);

// glPushMatrix();

// glTranslated(tX,tY,tZ);

// glScaled(1,1,2);

// //glRotated(90,1,0,0);

// glutSolidCube(1);

// glPopMatrix();

// glPopMatrix();

}

// Инициировать вызов функции рисования (объединяя компоненты вместе)

void draw() {

double t = glutGet(GLUT\_ELAPSED\_TIME) / 1000.0;

double a = t \* 90.0;

TIME = t;

/// Вызов самолета

if (rotX > 11)rotX = 11;

if (rotX < -11)rotX = -11;

if (rotZ > 10)rotZ = 10;

if (rotZ < -15)rotZ = -15;

glPushMatrix();

glTranslated(0, 1, 0);

glRotated(90, 0, 1, 0);

glRotated(5, 0, 0, 1);

glRotated(rotX, 1, 0, 0);

glRotated(rotY, 0, 1, 0);

glRotated(rotZ, 0, 0, 1);

glScaled(0.4, 0.4, 0.4);

plane();

glPopMatrix();

/// Вызов окружающей среде

if (tX >= 4.1)tX = 4.1;

if (tX <= -4.1)tX = -4.1;

if (tY > 0.1)tY = 0.1;

if (tY < -15)tY = -15;

glPushMatrix();

glTranslated(tX, tY, tZ);

environment(2);

glPopMatrix();

glPushMatrix();

glTranslated(tX, tY, tZ1);

soheedMinarEnv();

glPopMatrix();

glPushMatrix();

glTranslated(tX, tY, tZ2);

environment(3);

glPopMatrix();

glPushMatrix();

glTranslated(tX, tY, tZ3);

environment(1);

glPopMatrix();

glPushMatrix();

glTranslated(tX, tY, tZ4);

environment(5);

glPopMatrix();

glPushMatrix();

glTranslated(tX, tY, tZ5);

environment(4);

glPopMatrix();

glPushMatrix();

glTranslated(tX, tY, tZ6);

environment(2);

glPopMatrix();

// Конечно, независимо от того, как быстро движется самолет, среда будет преобразовываться соответствующим образом.

tZ += speed;

tZ1 += speed;

tZ2 += speed;

tZ3 += speed;

tZ4 += speed;

tZ5 += speed;

tZ6 += speed;

if (tZ >= 20)tZ = -110;

if (tZ1 >= 20)tZ1 = -110;

if (tZ2 >= 20)tZ2 = -110;

if (tZ3 >= 20)tZ3 = -110;

if (tZ4 >= 20)tZ4 = -110;

if (tZ5 >= 20)tZ5 = -110;

if (tZ6 >= 20)tZ6 = -110;

if (rotX > 0)rotX -= angleBackFrac;

if (rotX < 0)rotX += angleBackFrac;

if (rotY > 0)rotY -= angleBackFrac;

if (rotY < 0)rotY += angleBackFrac;

if (rotZ > 0)rotZ -= angleBackFrac;

if (rotZ < 0)rotZ += angleBackFrac;

//cout<<tX<<" "<<tY<<" "<<tZ<<endl;

//cout<<rotX<<" "<<rotY<<" "<<rotZ<<endl;

speed += 0.0002;

if (speed >= 0.7)speed = 0.7;

}

// Функция рисует текст посередине траектории полета самолета (с помощью пары команд glRasterPos и ​​glutBitmapCharacter) - стиль растрового изображения/\*

glRasterPos — Vị trí raster là vị trí trong tọa độ cửa sổ mà tại đó thao tác raster tiếp theo sẽ bắt đầu(giúp thực hiện viết texts trong openGL)

void glutBitmapCharacter(void \*font, int character);-hiển thị 1 kí tự bitmap trong openGL)

\*/

void drawBitmapText(const char \*str, float x, float y, float z)

{

const char \*c;

glRasterPos3f(x, y + 8, z);

for (c = str; \*c != '\0'; c++)

{

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10, \*c);

}

}

// Функция написания текста штрихами (светлая линия)

/\*

void glutStrokeCharacter(void \*font, int character);

\*/

void drawStrokeText(const char\* str, int x, int y, int z)

{

const char \*c;

glPushMatrix();

glTranslatef(x, y + 8, z);

glScalef(0.002f, 0.002f, z);

drawRain(); // поместите сюда, чтобы добавить к тексту эффект дождя

for (c = str; \*c != '\0'; c++)

{

glutStrokeCharacter(GLUT\_STROKE\_ROMAN, \*c);

}

glPopMatrix();

}

// Аналогично функции drawStrokeText.

void drawStrokeText2(const char\* str, int x, int y, int z)

{

const char \*c;

glPushMatrix();

glTranslatef(x, y + 8, z);

glScalef(0.005f, 0.005f, z);

for (c = str; \*c != '\0'; c++)

{

glutStrokeCharacter(GLUT\_STROKE\_ROMAN, \*c);

}

glPopMatrix();

}

// Функция для записи одного символа

void drawStrokeChar(char c, float x, float y, float z)

{

glPushMatrix();

glTranslatef(x, y + 8, z);

glScalef(0.002f, 0.002f, z);

glutStrokeCharacter(GLUT\_STROKE\_ROMAN, c);

glPopMatrix();

}

// Функция для представления символов, текста и количества пройденных круговых перекрестков

static void display(void)

{

const double t = glutGet(GLUT\_ELAPSED\_TIME) / 1000.0;

double a = t \* 90.0;

double aa = a;

//Nếu bỏ khối này đi là nhìn được xung quanh

if (!rot) {

a = 0;

}

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

glLoadIdentity();

gluLookAt(0.0, 4.5, 10.0,

0, 4, 0,

0, 1.0f, 0.0f);

if (START) {

glPushMatrix();

glTranslated(0, 0, 0);

glScaled(zoom, zoom, zoom);

glRotated(a, 0, 1, 0);

glBindTexture(GL\_TEXTURE\_2D, texture);

draw();

glPopMatrix();

drawStrokeText("UP: W, DOWN: S, LEFT: A, RIGHT: D, MAIN MENU: M", -8, 0.9, 0);

drawBitmapText("Quyet Tran Vu's OpenGL Planing Game Code", 0, 0, 0);

drawStrokeText("TIME : ", 3, 0, 0);

int mod, number = 0;

while (TIME) {

mod = TIME % 10;

number = number \* 10 + mod;

TIME /= 10;

}

//tmp là 1 giá trị để set up cho thời gian tăng lên đúng(tạo khoảng cách giữa các số)

float tmp = 0;

while (number) {

mod = number % 10;

//gọi hàm vẽ kí tự số(=số thời gian trôi qua)

drawStrokeChar(mod + 48, 4 + tmp, 0, 0);

number /= 10;

tmp += 0.2;

}

}

else {

glPushMatrix();

glTranslated(0, 3, 0);

glRotated(aa, 0, 1, 0);

glScaled(1.5, 1.5, 1.5);

plane();

glPopMatrix();

drawStrokeText("Press G to Start", -1, -1, 0);

drawStrokeText2("Plane Game", -2, 0, 0);

}

//glColor3d(1,1,0);

//drawStrokeText("Osama Hosam's OpenGL Tutorials",200,200,0);

// glRasterPos2i(100, 120);

// glColor4f(0.0f, 0.0f, 1.0f, 1.0f);

// glutBitmapString(GLUT\_BITMAP\_HELVETICA\_18, "text to render");

//drawStrokeChar(49,2,0,0);

glutSwapBuffers();

}

/\* Функция для обработки операций с мышью и клавиатурой : Имея в виду:

W: идти вверх

S: спуститься

D: вправо

A: слева

z: увеличить

Z: далеко

r: перейти в режим кросс-полета

t: вернуться в режим прямого полета\*/

static void key(unsigned char key, int x, int y)

{

float frac = 0.3;

float rotFrac = 1;

switch (key)

{

case 27:

case 'q':

exit(0);

break;

//hai trường hợp hỗ trợ đổi rot để thay đổi góc nhìn

case 'r':

rot = true;

break;

case 't':

rot = false;

break;

//hai trường hợp thay đổi hướng nhìn gần-xa

case 'z':

zoom += 0.05;

break;

case 'Z':

zoom -= 0.05;

//trường hợp di chuyển cho máy bay

case 'w':

tY -= frac;

rotZ += rotFrac;

break;

case 's':

tY += frac;

rotZ -= rotFrac;

break;

case 'a':

tX += frac;

rotX -= rotFrac \* 3;

rotY += rotFrac / 2;

break;

case 'd':

tX -= frac;

rotX += rotFrac \* 3;

rotY -= rotFrac / 2;

break;

// case 'y':

// rotX-=rotFrac;

// break;

// case 'h':

// rotX+=rotFrac;

// break;

// case 'g':

// rotY+=rotFrac;

// break;

// case 'j':

// rotY-=rotFrac;

// break;

//hai trường hợp bắt đầu,kết thúc game

case 'g':

START = true;

break;

case 'm':

START = false;

break;

// case 'o':

// cosX-=frac\*cos(rotX\*rad);

// cosY+=frac\*cos(rotY\*rad);

// cosZ-=frac\*cos(rotZ\*rad);

// //cout<<"Front : "<<cosX<<" "<<cosY<<" "<<cosZ<<endl;

// break;

// case 'l':

// cosX+=frac\*cos(rotX\*rad);

// cosY-=frac\*cos(rotY\*rad);

// cosZ+=frac\*cos(rotZ\*rad);

// //cout<<"Back : "<<cosX<<" "<<cosY<<" "<<cosZ<<endl;

// break;

}

glutPostRedisplay();

}

static void idle(void)

{

glutPostRedisplay();

}

// Использовать свет

const GLfloat light\_ambient[] = { 0.0f, 0.0f, 0.0f, 1.0f };

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

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

const GLfloat light\_position[] = { 2.0f, 5.0f, 5.0f, 0.0f };

const GLfloat mat\_ambient[] = { 0.7f, 0.7f, 0.7f, 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 };

/\* Program entry point \*/

/\* К основным функциям программы относятся:

-Начальная настройка, настройка свойств

-Воспроизведение аудио музыки (через функцию sndPlaySound, доступную в openGL)

-Включить режим разрешения атрибутов\*/

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

{

initRain();

glutInit(&argc, argv);

glutInitWindowPosition(0, 0);

glutInitWindowSize(1366, 720);

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

glutCreateWindow("GLUT Shapes-Fixed Project by Quyet");

glutReshapeFunc(resize);

glutDisplayFunc(display);

glutTimerFunc(0, timer, 0);

glutKeyboardFunc(key);

glutIdleFunc(idle);

//PlaySound(TEXT("[MP3DOWNLOAD.TO] Yiruma - River Flows in You-HQ.wav"), NULL, SND\_ASYNC|SND\_FILENAME|SND\_LOOP);

sndPlaySound("[MP3DOWNLOAD.TO] Yiruma - River Flows in You-HQ.wav", SND\_ASYNC);

glClearColor(1, 1, 1, 1);

glEnable(GL\_CULL\_FACE);// разрешить объектам смотреть вперед и назад (обратите внимание, если режим GL\_FRONT\_AND\_BACK: полигоны не отображаются, только точки и линии)

glCullFace(GL\_BACK);

glEnable(GL\_TEXTURE\_2D);

glEnable(GL\_DEPTH\_TEST);

// Используя GL\_LESS, я не буду смотреть на объекты "Пустой"

glDepthFunc(GL\_LESS);// По умолчанию функция глубины GL\_LESS используется для удаления всех сегментов со значением глубины больше или равным значению текущего буфера глубины.

glEnable(GL\_LIGHT0);

glEnable(GL\_NORMALIZE); // что позволяет нормализовать векторы после преобразования и до освещения.

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

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

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

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

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

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

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

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

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

glutMainLoop();

return EXIT\_SUCCESS;

}

Объяснить значение эффекта дождя:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include<glut.h>

#define RAINSIZE 1000 // переменная, определяющая плотность дождевых капель (чем она больше, тем четче выпадет количество дождя)

int winWidth = 1000, winHeight = 1000;

//int counter = 0;

//time\_t t;

float rotationAngle = 0; // атрибут, определяющий угол падения дождя

// создать структуру, содержащую атрибуты, относящиеся к капле дождя (координаты; движение; радиус; длина; угол падения; угол движения)

struct drop {

float x = 400;

float y = 400;

float inc = 0.01;

float radius = 10;

float scale = 1.0;

float rotationAngle = 0;

float rotationInc = 1;

};

drop rain[RAINSIZE];

// Используя функцию rand (), я создам любое значение

void initRain() {

//srand((unsigned)time(&t));

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

rain[i].x = rand() % winWidth;

rain[i].y = rand() % winHeight;

rain[i].inc = 1.5 + (float)(rand() % 100) / 1000.0;

rain[i].radius = (float)(rand() % 8);

rain[i].scale = (float)(rand() % 20000) / 1000.0;

rain[i].rotationAngle = (float)(rand() % 3000) / 1000.0;

rain[i].rotationInc = (float)(rand() % 100) / 1000.0;

if ((rand() % 100) > 1000) {

rain[i].rotationInc = -rain[i].rotationInc;

}

}

}

// Нарисуйте дождь (соедините 2 точки прямой линией) (обратите внимание, что я соединю 2 точки с координатами (x, y) и точкой (x, y + радиус со второй точкой))

void drawParticleShape(int i) {

glBegin(GL\_POINTS);

glVertex2d(rain[i].x, rain[i].y);

glEnd();

glBegin(GL\_LINES);

glVertex2d(rain[i].x, rain[i].y);

glVertex2d(rain[i].x, rain[i].y + rain[i].radius \* 2);

glEnd();

}

// Реализовать создание капли

void drawDrop(int i) {

glColor3f(0.0, 0.0, 1.0); //màu xanh lơ

glLineWidth(2);

drawParticleShape(i);

rain[i].y -= rain[i].inc; // каждый раз, когда дождь идет по всей длине, я вычитаю его из значения движения и переназначаю исходное значение высоты на каплю дождя.

if (rain[i].y < 0) {

rain[i].y = winHeight;

}

}

// Вызов функции дождя, используя цикл, включающий плотность дождя

void drawRain() {

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

drawDrop(i);

}

}