МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Чувашский государственный университет имени И.Н. Ульянова»

Факультет информатики и вычислительной техники

Кафедра вычислительной техники

Отчет по лабораторной работе №3

по дисциплине: «Алгоритмические основы компьютерной графики»

«Положение камеры (наблюдателя)»

Выполнил: студент гр. ИВТ-41-19

Гоглев Сергей Владимирович

Проверил:

Васильев Никита Сергеевич

Чебоксары 2021

**Задание к лабораторной работе:**

Лабораторная работа №3 строится на основе предыдущей работы с

внесением следующих изменений:

1. В отдельном модуле реализовать класс Camera вышеописанной структуры

для работы с направленной камерой. Камера должна выполнять

требования, описанные в соответствующем разделе. Для лучшего

понимания принципов передвижения камеры можно обратиться к

демонстрационному примеру.

2. Привести функцию Display к виду описанному выше. При необходимости в

функцию можно добавить дополнительные возможности, однако общий

принцип должен оставаться неизменным – функция Display служит только

для вывода изображения на экран и не должна модифицировать никакие

глобальные данные.

3. Реализовать функцию Simulation. Функция должна регистрироваться с

помощью glutIdleFunc, определять состояние требуемых клавиш, время

симуляции и вызывать соответствующие методы для изменения позиции камеры.

Текст заголовочного файла модуля с классом Camera.

#pragma once

#include <Windows.h>

#include "GL/freeglut.h"

#include "math.h"

class Camera

{

// необходимые свойства

float x, y, z, r, fi = 45, om = 45, time;

bool left, right, up, down, forward, backward;

public:

// конструктор по умолчанию

Camera(void);

// деструктор ‐ сохранение новых параметров камеры

~Camera(void);

// установка признаков нажатых клавиш

void setKey(bool left, bool right, bool up, bool down, bool forward, bool backward);

// движение камеры в ранее выбранном направлении

// параметр ‐ количество секунд прошедших с момента последнего вызова

void simulate(float sec);

// функция для установки матрицы камеры

void apply();

};

Текст файла реализации модуля с классом Camera.

// Вычисляется начальный (любой) радиус

Camera::Camera()

{

r = sqrt(444);

}

// Деструктор класса Camera

Camera::~Camera() {}

// Вычисляется кол-во секунд прошедших с момента последнего вызова

void Camera::simulate(float sec)

{

time = sec;

}

// Считывание нажатых клавиш

void Camera::setKey(bool left, bool right, bool up, bool down, bool forward, bool backward)

{

Camera::left = left; - если лево

Camera::right = right; - если право

Camera::up = up; - если вверх

Camera::down = down; - если вниз

Camera::forward = forward; - если вперед

Camera::backward = backward; - если назад

// Вычисляется угол поворота при различных нажатых клавишах, нужно для правильного перемещения камеры

if (left)

{

fi = fi + 90 \* time;

}

if (right)

{

fi = fi + (-90) \* time;

}

if (up)

{

om = min(om + 90 \* time, 90);

}

if (down)

{

if (om > 0)

{

om = om + (-90) \* time;

}

}

if (forward)

{

if (r > 8)

{

r = r + (-30) \* time;

}

}

if (backward)

{

if (r < 30)

{

r = r + 30 \* time;

}

}

}

// В данном методе выполняется итоговое вычисление координат

void Camera::apply()

{

// перевод из радианов в градусы (или наоборот)

float fiR = fi \* 0.017;

float omR = om \* 0.017;

// формула вычисления по сферической системе координат

x = r \* cos(fiR) \* cos(omR);

y = r \* sin(omR);

z = r \* sin(fiR) \* cos(omR);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

// изменение позиции нашей камеры (ее вращение)

gluLookAt(x, y, z, 0, 0, 0, 0, 1, 0);

}

Текст основной программы с комментариями.

// функция вызывается когда процессор простаивает, т.е. максимально часто

void Simulation()

{

// ОПРЕДЕЛЕНИЕ ВРЕМЕНИ ПРОШЕДШЕГО С МОМЕНТА ПОСЛЕДНЕЙ СИМУЛЯЦИИ В СЕКУНДАХ

float Simulation\_Time\_Passed = GetSimulationTime();

// ПЕРЕМЕЩЕНИЕ КАМЕРЫ

bool CameraLeft = GetAsyncKeyState(VK\_LEFT);

bool CameraRight = GetAsyncKeyState(VK\_RIGHT);

bool CameraUp = GetAsyncKeyState(VK\_UP);

bool CameraDown = GetAsyncKeyState(VK\_DOWN);

bool CameraForward = GetAsyncKeyState(VK\_ADD);

bool CameraBackward = GetAsyncKeyState(VK\_SUBTRACT);

camera.setKey(CameraLeft, CameraRight,

CameraUp, CameraDown,

CameraForward, CameraBackward);

camera.simulate(Simulation\_Time\_Passed);

// ПЕРЕРИСОВАТЬ ОКНО

glutPostRedisplay();

};

// Вычисление разницы тиков с начала работы программы и до обновления

double thisDelta(LARGE\_INTEGER& nTick, LARGE\_INTEGER& oTick)

{

return double(nTick.QuadPart - oTick.QuadPart) / frequency.QuadPart;

}

double getSimTime()

{

oldSimTick = newSimTick;

// возвращает текущее значение счетчика производительности реализуемого операционной системой

QueryPerformanceCounter(&newSimTick);

return thisDelta(newSimTick, oldSimTick);

}

// Функция вывода фпс

void outFps()

{

// возвращает текущее значение счетчика производительности реализуемого операционной системой

QueryPerformanceCounter(&newTick);

double delta = thisDelta(newTick, oldTick);

if (delta > 0.5) // изменение фпс каждый пол секунды

{

// вычисление фпс и перевод его в строку

std::string FPS = std::to\_string(counter / delta);

std::string newTitle = FPS + " FPS";

// вывод фпс вместо имени

glutSetWindowTitle(newTitle.c\_str());

counter = 0;

oldTick = newTick;

}

}

// инициализация библиотеки GLUT

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

{

/\* отрисока объектов \*/

/\* создание окна \*/

/\* всякие настройки \*/

// возвращает текущее значение счетчика производительности реализуемого операционной системой

QueryPerformanceCounter(&ticks);

// показывает с какой частотой меняется внутренний счетчик производительности

QueryPerformanceFrequency(&frequency);

// основной цикл обработки сообщений ОС

glutMainLoop();

}

Скриншот работы программы:

