КАФЕДРА №

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Работа со списками, текстурой и NURBS-поверхностями

по курсу: Компьютерная графика

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

1) Листинг программы:

Main.cpp

```
#include <iostream>
#include <cmath>

#include <GL/freeglut.h>

// функция для получения текстуры
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

#include "move.h" // функции для ПЕРЕДВИЖЕНИЯ
#include "models.h" // модели для списка
#include "draw.h" // функции для ПРОРИСОВКИ

#define TITLE "Nurbs, Textures && List"

#define W_WIDTH 1280
#define W_HEIGHT 720

#define FPS 60

namespace global {
float cam_xz_rotate = 1.5;
float cam_y_rotate = 0.4;
float cam_zoom = 20;
bool light_flag = true;
bool fog_flag = true;
}

// замена while
void timer(int value) {
    glutPostRedisplay();
    glutTimerFunc(1000/FPS, timer, 0);
}

// ну тут уже ОЧЕВИДНО
int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowPosition(150, 50);
    glutInitWindowSize(W_WIDTH, W_HEIGHT);
    glutCreateWindow(TITLE);

    glPointSize(5);

    init_surface();
    fog_on();
    generate_textures();
    generate_list();
```

```
    // передаём функции для прорисовки
    glutReshapeFunc(Reshape);
    glutDisplayFunc(Display);

    //
    glutSetKeyRepeat(GLUT_KEY_REPEAT_OFF);

    // передаём функции для определения нажатых клавиш
    glutKeyboardFunc(keyDown);
    glutKeyboardUpFunc(keyUp);
    glutMouseWheelFunc(MouseWheel);

    // таймер
    glutTimerFunc(0, timer, 0);

    // чтоб окно не закрывалось
    glutMainLoop();
}
```

Draw.h

```
/*
 Здесь реализуются функции для ПРОРИСОВКИ
 */

#include <iostream>
#include <GL/freeglut.h>

#include <cmath>

/*
 mat_dif - цвет рассеянного отражения материала (прозрачность задаётся 4 параметром)
 mat_spec - цвет зеркального отражения материала
 mat_amb - цвет фонового отражения материала
 mat_shininess - коэффициент блеска
 */

namespace global {
extern float cam_xz_rotate;
extern float cam_y_rotate;
extern float cam_zoom;
extern bool light_flag;
extern bool fog_flag;
}

// список
GLuint model_list;

// текстура
GLuint texture;
```

```
// параметры для nurb поверхности
GLUnurbsObj* theNurb;

const int size_numb_x = 4;
const int size_numb_y = 4;

GLfloat ctlpoints[size_numb_x][size_numb_y][3];
GLfloat knots[size_numb_x + size_numb_y] = {
   0.0, 0.0, 0.0, 0.0,
   1.0, 1.0, 1.0, 1.0
};

// материал nurb поверхности
float mat_dif_Nurb[] = {0.0f, 0.4f, 0.0f, 1.0f};
float mat_spec_Nurb[] = {0.0f, 0.4f, 0.0f};
float mat_amb_Nurb[] = {0.0f, 0.4f, 0.0f};
float mat_shininess_Nurb = 0.1f * 128;

// генерируем точки поверхности
void init_surface() {
   int m = 5; // расстояние между точками
   for (int y = 0; y < size_numb_y; y++) {
      for (int x = 0; x < size_numb_x; x++) {
         ctlpoints[y][x][0] = m * ((GLfloat)y - 1.5);
         ctlpoints[y][x][1] = m * ((GLfloat)x - 1.5);

         // выпуклости на поверхности
         if ((y == 1 ||y == 2) && (x == 1 || x == 2))
            ctlpoints[y][x][2] = 10.0;
      }
   }

   theNurb = gluNewNurbsRenderer();
   gluNurbsProperty(theNurb, GLU_SAMPLING_TOLERANCE, 5.0);
   gluNurbsProperty(theNurb, GLU_DISPLAY_MODE, GLU_FILL);
   gluNurbsProperty(theNurb, GLU_AUTO_LOAD_MATRIX, false);
}

// метериалы объектов
float mat_dif_list[] = {0.9f, 0.9f, 0.9f, 1.0f};
float mat_spec_list[] = {0.9f, 0.9f, 0.9f};
float mat_amb_list[] = {0.9f, 0.9f, 0.9f};
float mat_shininess_list = 0.1f * 128;

// настройки света
// материал объекта для визуализации источника
float mat_dif_light[] = {0.9f, 0.9f, 0.0f, 1.0f};
float mat_spec_light[] = {0.9f, 0.9f, 0.0f};
float mat_amb_light[] = {0.9f, 0.9f, 0.0f};
float mat_shininess_light = 0.1f * 128;
```

```
// материал света
float matl_dif_light[] = {0.9f, 0.9f, 0.9f, 1.0f};

// перемещение света
float light_position_fraction = 0.02;
float light_position_radius = 10;
float light_tick = 0;
// центр вращения XYZ и W (если поставить не 0, то будет точечный источник
света)
float light_position[3][4] = {
    {0.0, 5.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0},
    {0.0, 0.0, 0.0, 0.0}
};

// функция для отображения и включения света
void set_light(GLenum name, GLfloat *light_position) {
    // материал света
    glLightfv(name, GL_DIFFUSE, matl_dif_light);
    glLightfv(name, GL_POSITION, light_position);
    glLightf(name, GL_SPOT_CUTOFF, 360);

    // материал объекта который визуализирует источник света
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_amb_light);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_dif_light);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec_light);
    glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess_light);

    glTranslatef(light_position[0], light_position[1], light_position[2]);
    glutSolidSphere(0.2, 32, 32);
    glTranslatef(-light_position[0], -light_position[1], -light_position[2]);

    // включаем свет
    glEnable(name);
}

// чтение и преобразование текстур
void generate_textures() {
    int width, height, nrChannels;
    unsigned char *data;

    data = stbi_load("texture.png", &width, &height, &nrChannels, 0);
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, data);
}

// натсройки для тумана
GLfloat fogColor[4] = { 0.5f, 0.5f, 0.5f, 1.0f };
```

```
void fog_on() {
   glEnable(GL_FOG);                    // Включает туман (GL_FOG)
   glFogi(GL_FOG_MODE, GL_EXP);// Выбираем тип тумана
   glFogfv(GL_FOG_COLOR, fogColor);        // Устанавливаем цвет тумана
   glFogf(GL_FOG_DENSITY, 0.05f);          // Насколько густым будет туман
   glHint(GL_FOG_HINT, GL_DONT_CARE);      // Вспомогательная установка
тумана
   glFogf(GL_FOG_START, 1.0f);             // Глубина, с которой начинается туман
   glFogf(GL_FOG_END, 5.0f);               // Глубина, где туман заканчивается.
}

// генерация объекта для списка
void generate_list() {
   model_list = glGenLists(1);

   glNewList(model_list, GL_COMPILE);

   glEnable(GL_TEXTURE_2D);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
   model_cube();

   glScalef(0.5, 0.5, 0.5);
   glTranslatef(0, 0, 3);
   model_cube();

   glScalef(2, 2, 2);
   glTranslatef(0, 0, -1.5);

   glDisable(GL_TEXTURE_2D);

   glEndList();
}

void Display(void) {
   glLoadIdentity();

   // рассчитываем передвижение для камеры
   move();

   // задаём параметры камере
   gluLookAt(
         cos(global::cam_y_rotate) * sin(global::cam_xz_rotate) * global::cam_zoom,
         sin(global::cam_y_rotate) * global::cam_zoom,
         cos(global::cam_y_rotate) * cos(global::cam_xz_rotate) * global::cam_zoom,
         0, 0, 0,
         0.0f, 1.0f, 0.0f
         );

   // фон
   glClearColor(0.7, 0.7, 0.7, 1);
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
// без этого буквально все элементы становятся 3D (видно сквозь объекты)
glEnable(GL_DEPTH_TEST);

// включаем смешивание цвета (для прозрачности)
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

// туман
if (global::fog_flag) glDisable(GL_FOG);
else glEnable(GL_FOG);

/* включаем освещение */
glEnable(GL_LIGHTING);

// перемещение испочников света
if (global::light_flag) light_tick += light_position_fraction;
float light_position_rotate[4];

/* настриваем источники света */
// крутится вокруг сцены (XZ)
light_position_rotate[0] = light_position[0][0] + (cos(light_tick) * light_position_radius);
light_position_rotate[1] = light_position[0][1];
light_position_rotate[2] = light_position[0][2] + (sin(light_tick) * light_position_radius);
set_light(GL_LIGHT0, light_position_rotate);

// для того чтобы поверхность нормально реагировала на свет
glEnable(GL_AUTO_NORMAL);

// объекты
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_amb_Nurb);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_dif_Nurb);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec_Nurb);
glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess_Nurb);
glTranslatef(0.0f, 0.0f, 0.0f);
glRotatef(-90.0, 1.0, 0.0, 0.0);

// рисуем nurb поверхность
gluBeginSurface(theNurb);
gluNurbsSurface(theNurb,
            size_numb_x + size_numb_y, knots, size_numb_x + size_numb_y, knots,
            size_numb_x * 3, 3, &ctlpoints[0][0][0],
            4, 4, GL_MAP2_VERTEX_3);
gluEndSurface(theNurb);

glMaterialfv(GL_FRONT, GL_AMBIENT, mat_amb_list);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_dif_list);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec_list);
glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess_list);

glTranslatef(10.0f, -5.0f, 1.0f);
glCallList(model_list);
```

```
    glTranslatef(0.0f, 3.0f, 0.0f);
    glCallList(model_list);

    glTranslatef(0.0f, 3.0f, 0.0f);
    glCallList(model_list);

    glTranslatef(0.0f, 3.0f, 0.0f);
    glCallList(model_list);

    glPopMatrix();
    glFlush();
}

void Reshape(int w, int h) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(40.0, (GLfloat) w / h, 1, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
}
```

Models.h

```
/*
 Здесь храянтся модели
 */

#include <iostream>
#include <GL/freeglut.h>

void model_cube() {
    glBegin(GL_QUADS);

    glNormal3f(0.0, 1.0, 0.0);
    glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f,  1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, 1.0f,  1.0f);

    glNormal3f(0.0, -1.0, 0.0);
    glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);

    glNormal3f(0.0, 0.0, 1.0);
    glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f,  1.0f, 1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, 1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
```

```
    glNormal3f(0.0, 0.0, -1.0);
    glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f,  1.0f, -1.0f);

    glNormal3f(-1.0, 0.0, 0.0);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);

    glNormal3f(1.0f, 0.0f, 0.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(1.0f,  1.0f, -1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(1.0f,  1.0f,  1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, -1.0f,  1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, -1.0f, -1.0f);
    glEnd();
}
```

Move.h

```
/*
 Здесь реализуются функции для ПЕРЕДВИЖЕНИЯ
 */

#include <GL/freeglut.h>

namespace global {
extern float cam_xz_rotate;
extern float cam_y_rotate;
extern float cam_zoom;
extern bool light_flag;
extern bool fog_flag;
}

bool left = false;
bool right = false;
bool up = false;
bool down = false;

// отпущенные клавишы
void keyUp(unsigned char key, int xx, int yy) {
    switch (key) {
        case ('a'):
            left = false;
            break;

        case ('d'):
            right = false;
```

```
            break;

        case ('w'):
            up = false;
            break;

        case ('s'):
            down = false;
            break;

            // выход
        case 27:
            exit (0);
            break;
    }
}

// нажатые клавишы
void keyDown(unsigned char key, int xx, int yy) {
    switch (key) {
        case ('a'):
            left = true;
            break;

        case ('d'):
            right = true;
            break;

        case ('w'):
            up = true;
            break;

        case ('s'):
            down = true;
            break;

        case ('l'):
            global::light_flag = !global::light_flag;
            break;

        case ('f'):
            global::fog_flag = !global::fog_flag;
            break;
    }
}

float fraction = 0.05;
float cam_y_rotate_max = 1.5;
// функция для рассчётов передвижений
void move() {
    if (left) {
        global::cam_xz_rotate -= fraction;
```

```
    }

    if (right) {
        global::cam_xz_rotate += fraction;
    }

    if (up && (global::cam_y_rotate + fraction < cam_y_rotate_max)) {
        global::cam_y_rotate += fraction;
    }

    if (down && (global::cam_y_rotate - fraction > -cam_y_rotate_max)) {
        global::cam_y_rotate -= fraction;
    }

}

float fraction_zoom = 1.0;
// прокрутка колеса мыши
void MouseWheel(int button, int dir, int x, int y) {
    if (dir > 0) {
        global::cam_zoom -= fraction_zoom;
    } else {
        global::cam_zoom += fraction_zoom;
    }
}
```

2) Вывод: были изучены принципы работы со списками, текстурой и NURBS-поверхностями.