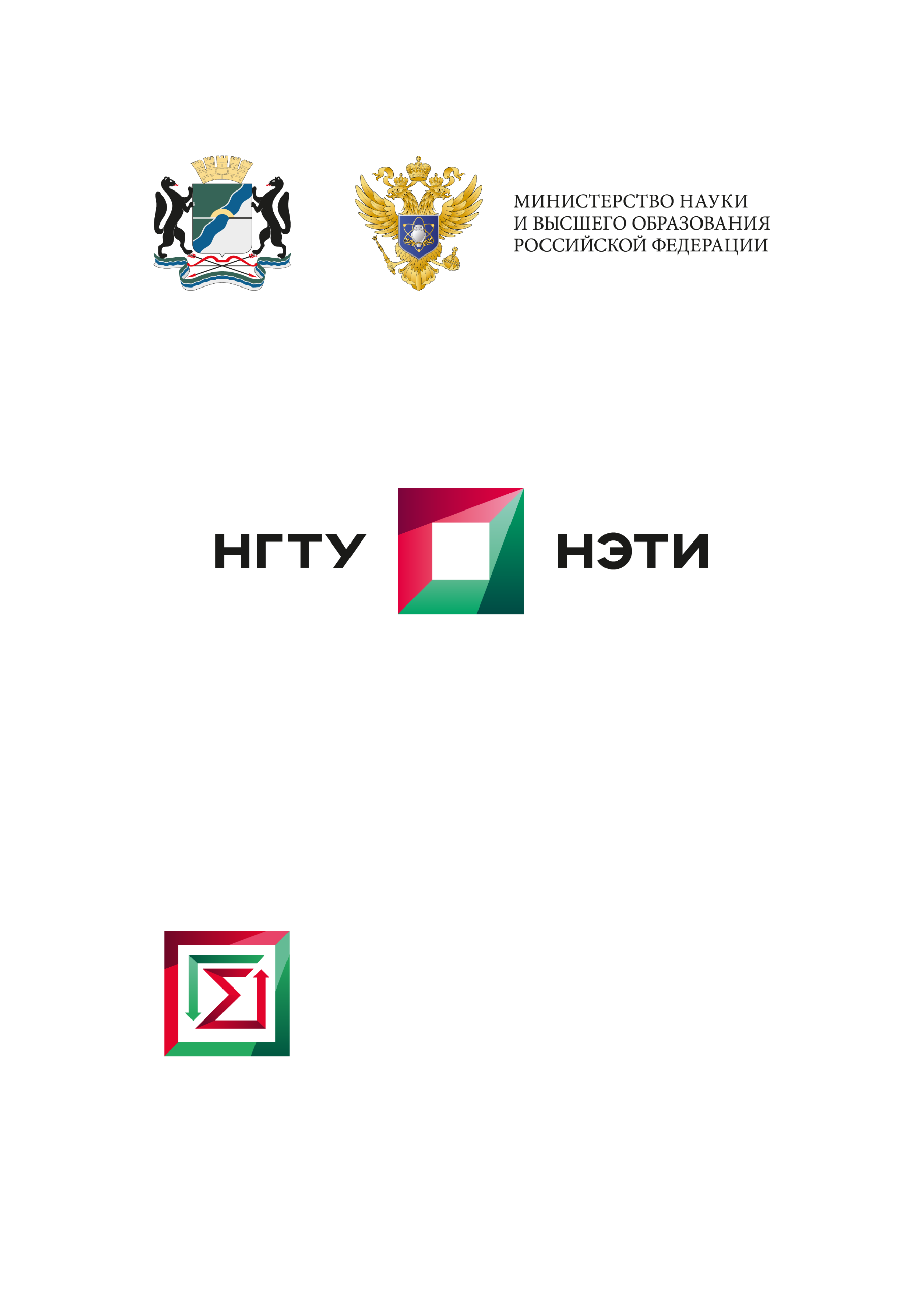
|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | | |
|  | | | |
| Кафедра прикладной математики | | | |
|  | | | |
| Лабораторная работа № 1 | | | |
| по дисциплине «Компьютерная графика» | | | |
| **Введение в программирование с использованием OpenGL** | | | |
|  | | | |
|  | Группа | ПМ-81 |
| Бригада | 14 |
| Студенты | Ефремов Артур |
|  | Редут Анатолий |
| Преподаватель | Задорожный А.Г. |
| Дата | 06.03.2021 |
|  | | | |
| Новосибирск | | | |



1. **Цель работы**

Ознакомиться с основами использования библиотеки OpenGL и работе с примитивами.

1. **Задание**
2. Отобразить в окне множество примитивов – незамкнутых ломаных (вершины которых задаются кликами мыши);

2. Для завершения текущего набора примитивов и начала нового зарезервировать специальную клавишу;

3. Предоставить возможность изменения цвета и координат вершин текущего набора примитивов;

1. Текущее множество примитивов выделять среди других;
2. Использовать контейнер vector из библиотеки STL для хранения набора примитивов и множества вершин каждого примитива, а для хранения атрибутов рекомендуется использовать стандартный класс struct;
3. Предусмотреть возможность удаления примитивов и наборов примитивов;
4. Продублировать команды в меню, созданном с помощью библиотеки GLUT.
5. **Руководство пользователя**

|  |  |  |
| --- | --- | --- |
| № | Кнопка мыши | Назначение |
| 1 | Левая | Добавление новой точки(по месту курсора) |
| 2 | Средняя | Удаление выделенной точки |

|  |  |  |
| --- | --- | --- |
| № | Клавиша | Назначение |
| 1 | Q, E | Поворот выделенной точки против часовой/по часовой оси |
| 2 | Стрелка влево, вправо | Переключение между точками в группе |
| 3 | F | Выделить центр/последнюю активную точку. Пересчитывает центр |
| 4 | Стрелка вниз, вверх, | Переключение между группами |
| 5 | T | Переключение режимов отрисовки |
| 6 | Z | Растянуть |
| 7 | X | Сжать |
| 8 | r,g,b | Изменение компоненты цвета |
| 9 | w, a, s, d | Перемещение выбранной точки или группы |
| 10 | C, V | Увеличить/уменьшить размер выделенных точек |
| 11 | +, - | Добавление/удаление группы |
| 12 | H | Переключение режимов сглаживания |

**Примечание:**

При выделенном центре клавиши перемещения перемещают всю группу.

Режимы сглаживания:

0 – нет сглаживания

1 – автоматически в зависимости от режима прорисовки.

2 – сглаживание точек и прямых

3 – сглаживание точек, прямых и полигонов

1. **Текст программы**

**Vec2.h**

#pragma once

class Vec2

{

public:

double x, y;

Vec2()

{

x = 0;

y = 0;

}

Vec2(const double& t\_x, const double& t\_y)

{

x = t\_x;

y = t\_y;

}

Vec2(const Vec2& vec)

{

x = vec.x;

y = vec.y;

}

Vec2& operator +=(const Vec2& rhs)

{

x += rhs.x;

y += rhs.y;

return \*this;

}

friend Vec2 operator +(Vec2 lhs, const Vec2& rhs)

{

lhs += rhs;

return lhs;

}

friend Vec2 operator -(Vec2 lhs, const Vec2& rhs)

{

lhs -= rhs;

return lhs;

}

Vec2& operator \*=(const double& rhs)

{

x \*= rhs;

y \*= rhs;

return \*this;

}

Vec2& operator /=(const double& rhs)

{

x /= rhs;

y /= rhs;

return \*this;

}

Vec2& operator -=(const Vec2& rhs)

{

x -= rhs.x;

y -= rhs.y;

return \*this;

}

friend Vec2 operator \*(Vec2 lhs, const double& rhs)

{

return lhs \*= rhs;

}

friend Vec2 operator /(Vec2 lhs, const double& rhs)

{

return lhs /= rhs;

}

}

**Point.h**

#pragma once

#include "Vec2.h"

class Point

{

private:

public:

Vec2 loc;

Point()

{

loc = Vec2();

}

Point(const double& t\_x, const double& t\_y)

{

loc = Vec2(t\_x, t\_y);

}

Point(const Vec2& vec)

{

loc = vec;

}

void DrawCasing(const double& size)

{

double scale = 0.75;

glBegin(GL\_LINE\_LOOP);

glVertex2i(loc.x + size \* scale, loc.y + size \* scale);

glVertex2i(loc.x - size \* scale, loc.y + size \* scale);

glVertex2i(loc.x - size \* scale, loc.y - size \* scale);

glVertex2i(loc.x + size \* scale, loc.y - size \* scale);

glEnd();

}

void Move(Vec2 t\_loc)

{

loc += Vec2(t\_loc.x, t\_loc.y);

}

void Rotate(const double& angle, Vec2 around)

{

Move(around \* -1);

loc.x = loc.x \* cos(angle) - loc.y \* sin(angle);

loc.y = loc.x \* sin(angle) + loc.y \* cos(angle);

Move(around);

}

void Scale(const double& factor, Vec2 from)

{

Move(from \* -1);

loc \*= factor;

Move(from);

}

};

**Group.h**

#pragma once

#include <vector>

#include "glut.h"

#include "Point.h"

using namespace std;

class Group

{

public:

vector<Point> points;

Point center;

GLubyte R = 0, G = 0, B = 0;

GLfloat size = 10;

int active\_point = -1;

bool is\_center\_active = false;

bool is\_smoothing = false;

int counter = 0;

int mode = 0;

Group()

{

}

// Вычисление геометрического центра группы

void CalcCenter()

{

center = Point(0, 0);

for(size\_t i = 0; i < points.size(); i++)

center.loc += points[i].loc;

center.loc /= points.size();

}

void ChoseNextActivePoint()

{

if(!is\_center\_active)

if(points.size())

active\_point = (active\_point + 1) % points.size();

}

void ChosePrevActivePoint()

{

if(!is\_center\_active)

if(points.size())

active\_point = (active\_point - 1 + points.size()) % points.size();

}

// Добавление точки в группу

void AddPoint(Point point)

{

if(points.size())

points.insert(points.begin() + active\_point + 1, point);

else

points.push\_back(point);

ChoseNextActivePoint();

CalcCenter();

}

// Удаление точки из группы

void DeletePoint()

{

if(points.size())

{

points.erase(points.begin() + active\_point);

CalcCenter();

}

ChosePrevActivePoint();

/\*if(points.size())

active\_point = points.size() - 1;

else

active\_point = -1;\*/

}

// Отрисовка точек в соответствии с выбранным алгоритмом

void Draw()

{

OnEnableSmoothing(counter);

glColor3ub(R, G, B);

glPointSize(size);

glBegin(GLenum(mode));

for(size\_t i = 0; i < points.size(); i++)

glVertex2i(points[i].loc.x, points[i].loc.y);

glEnd();

OnDisableSmoothing();

}

void DrawCenter()

{

glColor3ub(0, 0, 0);

glPointSize(size / 2);

glBegin(GL\_POINTS);

glVertex2i(center.loc.x + size / 2, center.loc.y);

glVertex2i(center.loc.x - size / 2, center.loc.y);

glVertex2i(center.loc.x, center.loc.y + size / 2);

glVertex2i(center.loc.x, center.loc.y - size / 2);

glVertex2i(center.loc.x, center.loc.y);

glEnd();

}

void DrawCasing()

{

glColor3ub(0, 0, 0);

glPointSize(size);

for(size\_t i = 0; i < points.size(); i++)

points[i].DrawCasing(size \* 1.2);

if(points.size())

{

if(is\_center\_active)

center.DrawCasing(size \* 1.5);

else

points[active\_point].DrawCasing(size \* 1.5);

}

}

void Move(const double& x, const double& y)

{

if (is\_center\_active)

{

for (size\_t i = 0; i < points.size(); i++)

points[i].Move(Vec2(x, y));

center.loc += Vec2(x, y);

}

else if (points.size())

{

points[active\_point].Move(Vec2(x, y));

CalcCenter();

}

}

/\*void Translate(Vec2 around)

{

for(size\_t i = 0; i < points.size(); i++)

points[i].loc += Vec2(around.x, around.y);

}\*/

void Rotate(const double& angle, Vec2 around)

{

if(is\_center\_active)

for(size\_t i = 0; i < points.size(); i++)

points[i].Rotate(angle, around);

else if(points.size())

points[active\_point].Rotate(angle, around);

}

void Scale(const double& factor, Vec2 from)

{

if(is\_center\_active)

for(size\_t i = 0; i < points.size(); i++)

points[i].Scale(factor, from);

else if(points.size())

points[active\_point].Scale(factor, from);

}

void OnEnableSmoothing(int counter)

{

if (is\_smoothing)

{

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glEnable(GL\_BLEND);

switch (counter)

{

case(1):

{

switch (mode)

{

case(0):

{

glEnable(GL\_POINT\_SMOOTH);

glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST);

break;

}

case(1):

case(2):

case(3):

case(7):

case(8):

{

glEnable(GL\_LINE\_SMOOTH);

glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST);

break;

}

case(4):

case(5):

case(6):

case(9):

case(10):

{

glEnable(GL\_POLYGON\_SMOOTH);

glHint(GL\_POLYGON\_SMOOTH\_HINT, GL\_NICEST);

break;

}

default:

break;

}

break;

}

case(2):

{

glEnable(GL\_POINT\_SMOOTH);

glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST);

glEnable(GL\_LINE\_SMOOTH);

glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST);

break;

}

case(3):

{

glEnable(GL\_POINT\_SMOOTH);

glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST);

glEnable(GL\_LINE\_SMOOTH);

glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST);

glEnable(GL\_POLYGON\_SMOOTH);

glHint(GL\_POLYGON\_SMOOTH\_HINT, GL\_NICEST);

break;

}

default:

break;

}

}

}

void OnDisableSmoothing()

{

glDisable(GL\_BLEND);

glDisable(GL\_POINT\_SMOOTH);

glDisable(GL\_LINE\_SMOOTH);

glDisable(GL\_POLYGON\_SMOOTH);

is\_smoothing = false;

}

};

**main.cpp**

#include <iostream>

#include <string>

#include "glut.h"

#include "Group.h"

#include "Point.h"

#include "Vec2.h"

#include <Windows.h>

using namespace std;

int WIDTH = 1000, HEIGHT = 500;

int active\_group = -1;

bool is\_action = false;

vector<Group> groups;

enum keys { Empty, KeyR, KeyG, KeyB, KeyW, KeyA, KeyS, KeyD, KeyQ, KeyE,

KeyZ, KeyX, KeyC, KeyV, KeyF, KeyT, KeyH, KeyPlus, KeyMinus,

KeyUp, KeyDown, KeyLeft, KeyRight

};

// Функция для отрисовки строки

void DrawString(int x, int y, GLubyte col, string s)

{

glColor3f(col, col, col);

glPushMatrix();

glTranslatef(x, y, 0);

double fac = 0.15;

glScalef(fac, fac, fac);

for(size\_t i = 0; i < s.size(); i++)

glutStrokeCharacter(GLUT\_STROKE\_ROMAN, s[i]);

glPopMatrix();

}

// Функция вывода на экран

void Display()

{

glClearColor(255, 255, 255, 1);

glClear(GL\_COLOR\_BUFFER\_BIT);

// Отрисовка точек всех групп

for(size\_t i = 0; i < groups.size(); i++)

groups[i].Draw();

// Отрисовка указателей активной группы

if(groups.size())

groups[active\_group].DrawCasing();

if(groups.size() && groups[active\_group].points.size() > 1)

groups[active\_group].DrawCenter();

string s;

s = "Total groups: " + to\_string(groups.size());

DrawString(10, 70, 200, s);

s = active\_group >=0 ? "Current group: " + to\_string(active\_group + 1) : "Current group: none";

DrawString(10, 50, 200, s);

if(active\_group != -1)

{

s = "Points in group: " + to\_string(groups[active\_group].points.size());

DrawString(10, 30, 200, s);

}

if(active\_group >= 0)

{

if(groups[active\_group].active\_point == -2)

{

s = "Current point: center";

DrawString(10, 10, 200, s);

}

else

if(groups[active\_group].active\_point != -1)

{

s = "Current point: " + to\_string(groups[active\_group].active\_point + 1);

DrawString(10, 10, 200, s);

}

}

glFinish();

}

// Функция изменения размеров окна

void Reshape(GLint w, GLint h)

{

WIDTH = w; HEIGHT = h;

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0, w, 0, h, -1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

// Функция обработки сообщений от клавиатуры 1

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

{

double move\_speed = 5;

double color\_speed = 5;

double size\_speed = 1;

double angle = 0.02;

double scale = 1.05;

if (active\_group != -1)

{

switch (key)

{

// Передвижение точек по осям

case 'w': groups[active\_group].Move(move\_speed \* +0, move\_speed \* +1); break;

case 'a': groups[active\_group].Move(move\_speed \* -1, move\_speed \* +0); break;

case 's': groups[active\_group].Move(move\_speed \* +0, move\_speed \* -1); break;

case 'd': groups[active\_group].Move(move\_speed \* +1, move\_speed \* +0); break;

// Вращение точек

case 'q': groups[active\_group].Rotate(angle, groups[active\_group].center.loc);

break;

case 'e': groups[active\_group].Rotate(-angle, groups[active\_group].center.loc);

break;

// Масштабирование точек

case 'z': groups[active\_group].Scale(scale, groups[active\_group].center.loc); break;

case 'x': groups[active\_group].Scale(1 / scale, groups[active\_group].center.loc); break;

// Изменение цвета точек

case 'r': groups[active\_group].R += color\_speed; break;

case 'g': groups[active\_group].G += color\_speed; break;

case 'b': groups[active\_group].B += color\_speed; break;

// Изменение размера точек

case 'c': groups[active\_group].size += size\_speed; break;

case 'v': groups[active\_group].size -= size\_speed; break;

// Добавление и удаление групп

case '=':

groups.push\_back(Group());

active\_group = groups.size() - 1;

break;

case '-':

if (groups.size())

groups.erase(groups.begin() + active\_group);

active\_group = groups.size() - 1;

break;

// Выбор центральной точки группы

case 'f':

if (groups.size())

{

if (groups[active\_group].points.size() > 1)

groups[active\_group].is\_center\_active = !groups[active\_group].is\_center\_active;

groups[active\_group].CalcCenter();

}

break;

// Выбор режима отрисовки

case 't':

if (groups.size())

groups[active\_group].mode = (groups[active\_group].mode + 1) % 10;

break;

case 'h':

if (groups.size())

if (groups[active\_group].is\_smoothing && groups[active\_group].counter > 3)

{

//groups[active\_group].OnDisableSmoothing();

groups[active\_group].counter = 0;

}

else

{

groups[active\_group].counter = (groups[active\_group].counter + 1) % 4;

cout << groups[active\_group].counter;

groups[active\_group].is\_smoothing = true;

//groups[active\_group].OnEnableSmoothing(counter);

}

break;

}

}

glutPostRedisplay();

}

// Функция обработки сообщений от клавиатуры 2

void KeyboardSpecials(int key, int x, int y)

{

double speed = 5;

switch(key)

{

// Переключение между активными группами

case GLUT\_KEY\_UP:

active\_group = (active\_group + 1) % groups.size();

if(groups.size())

groups[active\_group].is\_center\_active = true;

break;

case GLUT\_KEY\_DOWN:

active\_group = (active\_group - 1 + groups.size()) % groups.size();

if(groups.size())

groups[active\_group].is\_center\_active = true;

break;

// Переключение между точками активной группы

case GLUT\_KEY\_RIGHT:

if(active\_group != -1)

groups[active\_group].ChoseNextActivePoint();

break;

case GLUT\_KEY\_LEFT:

if(active\_group != -1)

groups[active\_group].ChosePrevActivePoint();

break;

}

glutPostRedisplay();

}

// Функция обработки сообщения от мыши

void Mouse(int button, int state, int x, int y)

{

// Клавиша была нажата, но не отпущена

if(state != GLUT\_DOWN) return;

// Новая точка по левому клику

if(button == GLUT\_LEFT\_BUTTON)

{

if(!groups.size())

{

groups.push\_back(Group());

active\_group = 0;

}

groups[active\_group].AddPoint(Point((double)x, (double)(HEIGHT - y)));

}

// Удаление последней точки по правому клику

if(button == GLUT\_MIDDLE\_BUTTON)

{

if(groups.size())

groups[active\_group].DeletePoint();

}

glutPostRedisplay();

}

void Menu(int pos)

{

int key = (keys)pos;

switch (key)

{

case KeyR: KeyboardLetters('r', 0, 0); break;

case KeyG: KeyboardLetters('g', 0, 0); break;

case KeyB: KeyboardLetters('b', 0, 0); break;

case KeyW: KeyboardLetters('w', 0, 0); break;

case KeyS: KeyboardLetters('s', 0, 0); break;

case KeyA: KeyboardLetters('a', 0, 0); break;

case KeyD: KeyboardLetters('d', 0, 0); break;

case KeyQ: KeyboardLetters('q', 0, 0); break;

case KeyE: KeyboardLetters('e', 0, 0); break;

case KeyC: KeyboardLetters('c', 0, 0); break;

case KeyV: KeyboardLetters('v', 0, 0); break;

case KeyZ: KeyboardLetters('z', 0, 0); break;

case KeyX: KeyboardLetters('x', 0, 0); break;

case KeyF: KeyboardLetters('f', 0, 0); break;

case KeyT: KeyboardLetters('t', 0, 0); break;

case KeyH: KeyboardLetters('h', 0, 0); break;

case KeyPlus: KeyboardLetters('=', 0, 0); break;

case KeyMinus: KeyboardLetters('-', 0, 0); break;

case KeyLeft: KeyboardSpecials(GLUT\_KEY\_LEFT, 0, 0); break;

case KeyRight: KeyboardSpecials(GLUT\_KEY\_RIGHT, 0, 0); break;

case KeyUp: KeyboardSpecials(GLUT\_KEY\_UP, 0, 0); break;

case KeyDown: KeyboardSpecials(GLUT\_KEY\_DOWN, 0, 0); break;

default:

int menu\_color = glutCreateMenu(Menu);

glutAddMenuEntry("Компонента R", KeyR);

glutAddMenuEntry("Компонента G", KeyG);

glutAddMenuEntry("Компонента B", KeyB);

int menu\_move = glutCreateMenu(Menu);

glutAddMenuEntry("Вверх", KeyW);

glutAddMenuEntry("Вниз", KeyS);

glutAddMenuEntry("Bлево", KeyA);

glutAddMenuEntry("Вправо", KeyD);

int menu\_size = glutCreateMenu(Menu);

glutAddMenuEntry("Увеличить", KeyC);

glutAddMenuEntry("Уменьшить", KeyV);

int menu\_scale = glutCreateMenu(Menu);

glutAddMenuEntry("Расстянуть", KeyZ);

glutAddMenuEntry("Сжать", KeyX);

int menu\_rotate = glutCreateMenu(Menu);

glutAddMenuEntry("Повернуть против часовой", KeyQ);

glutAddMenuEntry("Повернуть по часовой", KeyE);

int menu\_group = glutCreateMenu(Menu);

glutAddMenuEntry("Добавить группу", KeyPlus);

glutAddMenuEntry("Удалить группу", KeyMinus);

int menu\_misc = glutCreateMenu(Menu);

glutAddMenuEntry("Выбрать центр", KeyF);

glutAddMenuEntry("Переключение режима отрисовки", KeyT);

glutAddMenuEntry("Переключение режима сглаживания", KeyH);

int menu\_select = glutCreateMenu(Menu);

glutAddMenuEntry("Выбрать следующую вершину группы", KeyRight);

glutAddMenuEntry("Выбрать предыдущую вершину группы", KeyLeft);

glutAddMenuEntry("Выбрать следующую группу", KeyUp);

glutAddMenuEntry("Выбрать предыдущую группу", KeyDown);

int menu = glutCreateMenu(Menu);

glutAddSubMenu("Смена цвета", menu\_color);

glutAddSubMenu("Изменение размера точки", menu\_size);

glutAddSubMenu("Перемещение", menu\_move);

glutAddSubMenu("Масштабирование", menu\_scale);

glutAddSubMenu("Вращение", menu\_rotate);

glutAddSubMenu("Управление группами", menu\_group);

glutAddSubMenu("Переключение между точками и группами", menu\_select);

glutAddSubMenu("Дополнительные", menu\_misc);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

KeyboardLetters(Empty, 0, 0);

}

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB);

glutInitWindowSize(WIDTH, HEIGHT);

glutCreateWindow("КГ Лабораторная работа №1");

Menu(Empty);

glutDisplayFunc(Display);

glutReshapeFunc(Reshape);

glutKeyboardFunc(KeyboardLetters);

glutSpecialFunc(KeyboardSpecials);

glutMouseFunc(Mouse);

glutMainLoop();

}

1. **Тестирование**

|  |  |  |
| --- | --- | --- |
| № | Кнопка мыши | Назначение |
| 1 | Левая | Добавление новой точки(по месту курсора) |
| 2 | Средняя | Удаление выделенной точки |

|  |  |  |
| --- | --- | --- |
| № | Клавиша | Назначение |
| 1 | Q, E | Поворот выделенной точки против часовой/по часовой оси |
| 2 | Стрелка влево, вправо | Переключение между точками в группе |
| 3 | F | Выделить центр/последнюю активную точку. Пересчитывает центр |
| 4 | Стрелка вниз, вверх, | Переключение между группами |
| 5 | T | Переключение режимов отрисовки |
| 6 | Z | Растянуть |
| 7 | X | Сжать |
| 8 | r,g,b | Изменение компоненты цвета |
| 9 | w, a, s, d | Перемещение выбранной точки или группы |
| 10 | C, V | Увеличить/уменьшить размер выделенных точек |
| 11 | +, - | Добавление/удаление группы |
| 12 | H | Переключение режимов сглаживания |

|  |
| --- |
| Добавление новой точки |
|  |