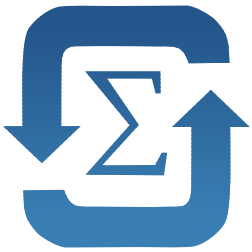
Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики

Лабораторная работа №1
по дисциплине «Комьютерная графика»

Введение в программирование с использованием OpenGL



Факультет: ПМИ

Группа: ПМ-63

Студент: Шепрут И.И.

Вариант: 9

Преподаватель: Задорожный А.Г.

Новосибирск
2019

1 Цель работы

Ознакомиться с основами использования библиотеки OpenGL и работе с примитивами.

2 Постановка задачи

1. Отобразить в окне множество примитивов (вершины которых задаются кликами мыши) в соответствии с вариантом задания.
2. Для завершения текущего (активного) набора (множества) примитивов и начала нового зарезервировать специальную клавишу (пробел или правый клик).
3. Для текущего набора примитивов предоставить возможность изменения цвета и координат его вершин.
4. Текущее множество примитивов выделять среди других, например, изменением размера его вершин командой `glPointSize(*)`.
5. Использовать контейнер `vector` из библиотеки `STL` для хранения набора примитивов и множества вершин каждого примитива, а для хранения атрибутов рекомендуется использовать стандартный класс `struct`.
6. Предусмотреть возможность удаления последнего примитива и последнего набора примитивов.
7. Продублировать команды в меню, созданном с помощью библиотеки `GLUT`.
   Вариант: `9. GL_QUAD_STRIP`.

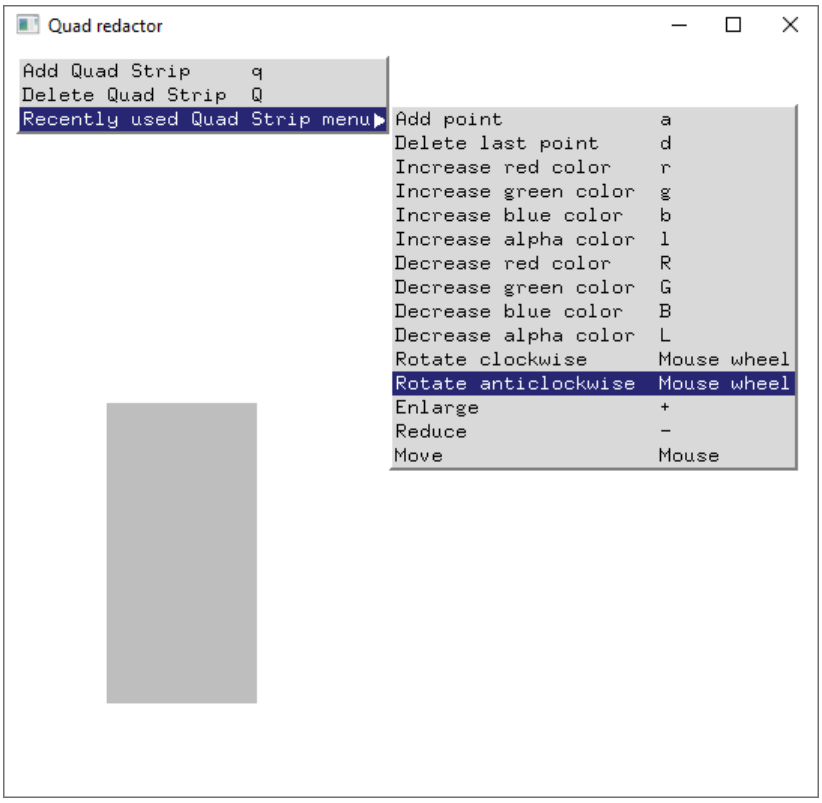
3 Реализованные функции

- **Выделение произвольного примитива как текущий с помощью мыши.**
- Добавление, удаление примитива.
- Операции над текущим примитивом:
  - Добавление, удаление точки.
  - **Изменение координаты произвольной точки с помощью мыши.**
  - Изменение цвета примитива (в том числе и альфа-канала).
  - Масштабирование примитива.
  - **Вращение примитива вокруг мыши.**
- Все функции продублированы в меню, там же написана клавиша, реализующая эту функцию.
- При рисовании используется двойная буферизация.

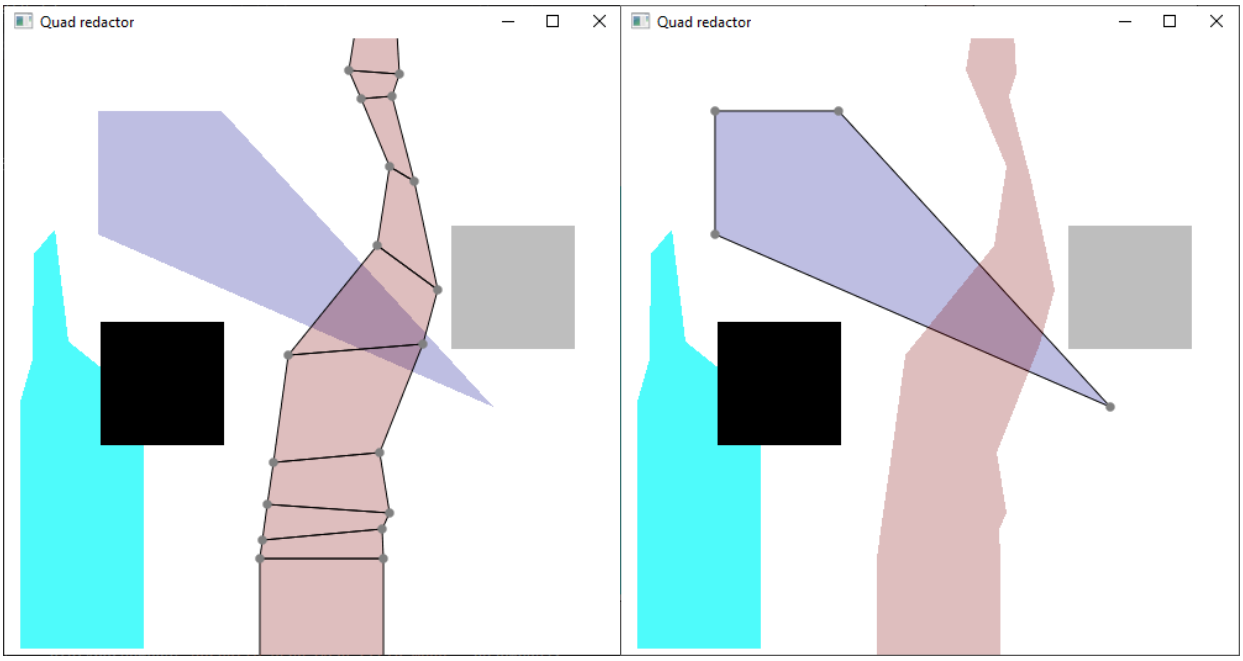
4 Пример реализованных функций

4.1 Меню

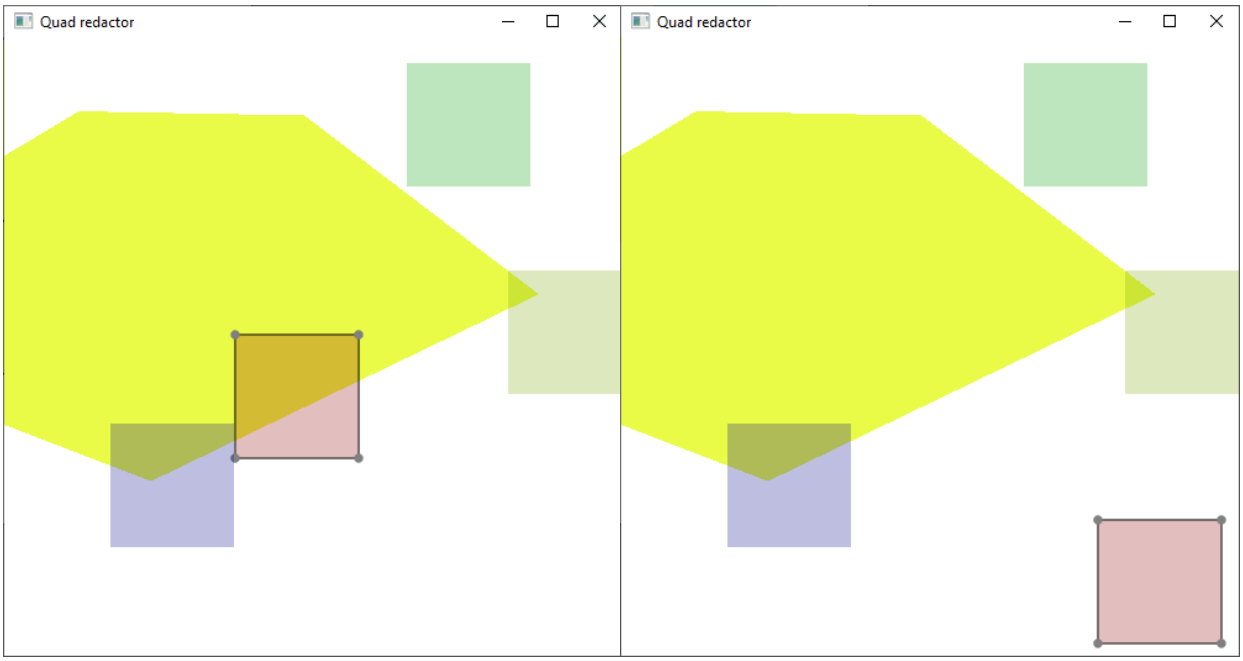
В меню так же с отступом написано как воспользоваться данной возможностью при помощи клавиатуры, а конкретно, там сказана клавиша, реализующая это действие. Так же там говорится какие действия выполняются с помощью мыши и колесика мыши.



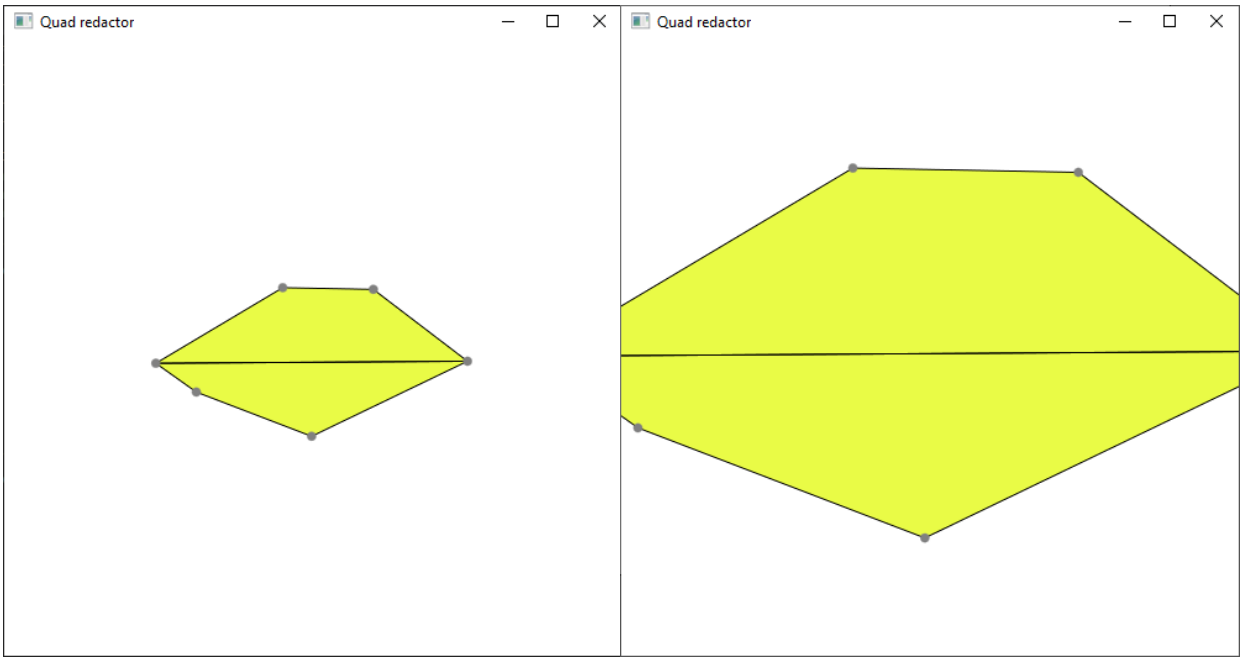
4.2 Выбор произвольного примитива с помощью мыши



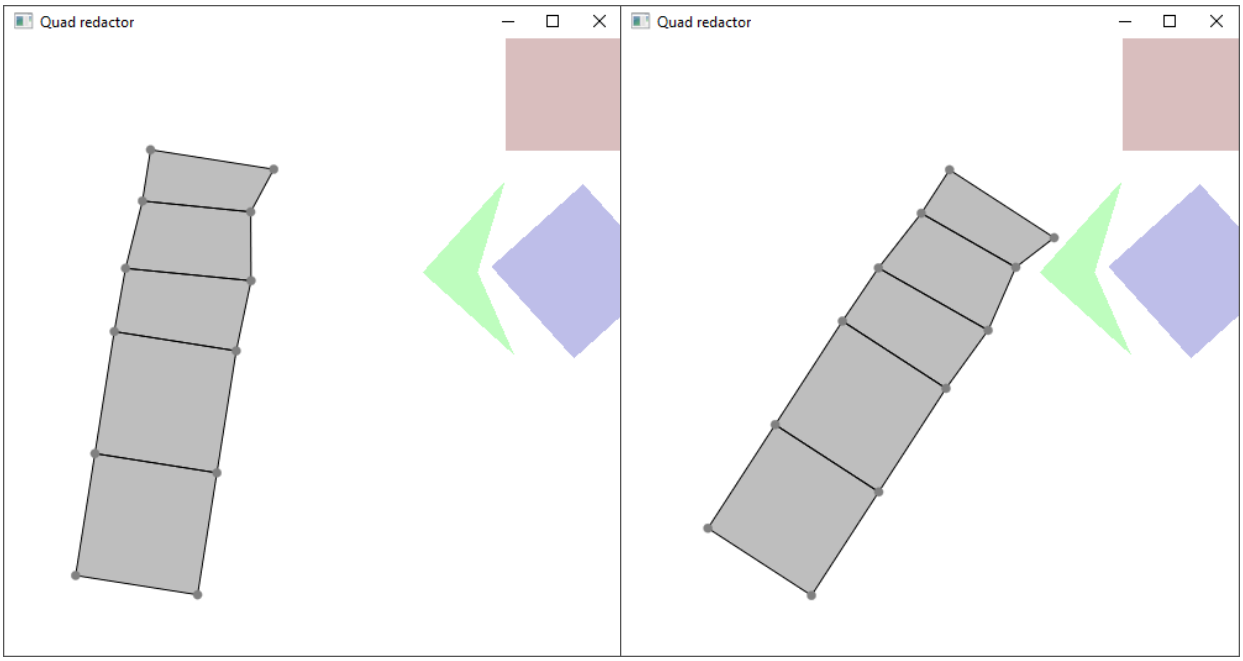
4.3 Перемещение примитива



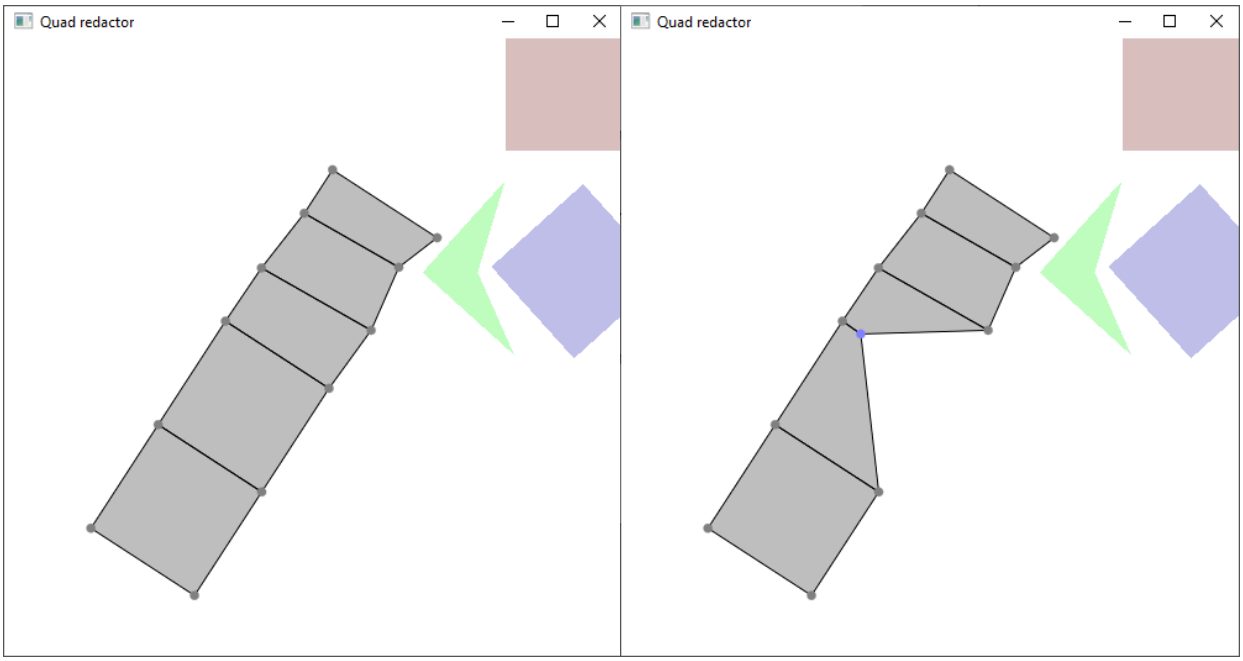
4.4 Масштабирование примитива



4.5 Вращение текущего примитива вокруг мыши



4.6 Изменение произвольной точки примитива



5 Код программы

FILE **main.cpp**

```
#include <iostream>
#include <vector>
#include <string>
#include <GL/freeglut.h>
#include ”interface.h”

std::vector<QuadStripInterface> quads;
```

```
int lastUsed = 0;
int Width = 500, Height = 500;
int lastX = 0, lastY = 0;

//-----------------------------------------------------------------------------
void Display(void) {
    glClearColor(1, 1, 1, 1);
    glClear(GL_COLOR_BUFFER_BIT);

    for (auto& i : quads)
        i.display();

    glutSwapBuffers();
}

//-----------------------------------------------------------------------------
void Reshape(GLint w, GLint h) {
    Width = w; Height = h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, w, 0, h);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

//-----------------------------------------------------------------------------
void Keyboard(unsigned char key, int x, int y) {
    lastX = x; lastY = y;
    y = Height - y;
    vec2 pos(x, y);
    int j = 0;
    for (auto& i : quads) {
        if (i.processKeyboard(key, pos)) { lastUsed = j; break; }
        j++;
    }

    //-------------------------------------------------------------------------
    // Добавляем или удаляем точку у последнего использованного объекта
    if (key == 'a')
        quads[lastUsed].points.push_back(PointInterface(pos, 4));

    if (key == 'd')
        if (quads[lastUsed].points.size() > 1)
            quads[lastUsed].points.pop_back();

    //-------------------------------------------------------------------------
    // Добавляем новый объект, или удаляем последний использованный
    if (key == 'q') {
        QuadStripInterface q;
        q.points.push_back(PointInterface(pos, 4));
        q.points.push_back(PointInterface(pos+vec2(100, 0), 4));
        q.points.push_back(PointInterface(pos+vec2(0, 100), 4));
        q.points.push_back(PointInterface(pos+vec2(100, 100), 4));
        quads.push_back(q);
    }

    if (key == 'Q') {
        if (quads.size() > 1) {
```

```
            quads.erase(quads.begin() + lastUsed);
            if (lastUsed == quads.size()) lastUsed--;
        }
    }

    glutPostRedisplay();
}

//----------------------------------------------------------------------------
void Mouse(int button, int state, int x, int y) {
    lastX = x; lastY = y;
    y = Height - y;
    vec2 pos(x, y);
    int j = 0;
    for (auto& i : quads) {
        if (i.processMouse(button, state, pos)) { lastUsed = j; break; }
        j++;
    }
    glutPostRedisplay();
}

//----------------------------------------------------------------------------
void Motion(int x, int y) {
    lastX = x; lastY = y;
    y = Height - y;
    vec2 pos(x, y);
    int j = 0;
    for (auto& i : quads) {
        if (i.processMotion(pos)) { lastUsed = j; break; }
        j++;
    }
    glutPostRedisplay();
}

//----------------------------------------------------------------------------
void PassiveMotion(int x, int y) {
    lastX = x; lastY = y;
    y = Height - y;
    vec2 pos(x, y);
    int j = 0;
    for (auto& i : quads) {
        if (i.processPassiveMotion(pos)) { lastUsed = j; break; }
        j++;
    }
    glutPostRedisplay();
}

//----------------------------------------------------------------------------
void MouseWheel(int button, int dir, int x, int y) {
    lastX = x; lastY = y;
    y = Height - y;
    vec2 pos(x, y);
    int j = 0;
    for (auto& i : quads) {
        if (i.processWheel(button, dir, pos)) { lastUsed = j; break; }
        j++;
    }
    glutPostRedisplay();
}
```

```
126
127 //---------------------------------------------------------------------------
128 void menu(int num) {
129     switch (num) {
130         case 1:  Keyboard('q', lastX, lastY);     break;
131         case 2:  Keyboard('Q', lastX, lastY);     break;
132         case 3:  Keyboard('a', lastX, lastY);     break;
133         case 4:  Keyboard('d', lastX, lastY);     break;
134         case 5:  Keyboard('r', lastX, lastY);     break;
135         case 6:  Keyboard('g', lastX, lastY);     break;
136         case 7:  Keyboard('b', lastX, lastY);     break;
137         case 8:  Keyboard('l', lastX, lastY);     break;
138         case 9:  Keyboard('R', lastX, lastY);     break;
139         case 10: Keyboard('G', lastX, lastY);     break;
140         case 11: Keyboard('B', lastX, lastY);     break;
141         case 12: Keyboard('L', lastX, lastY);     break;
142         case 13: MouseWheel(0, -1, lastX, lastY);  break;
143         case 14: MouseWheel(0, 1, lastX, lastY); break;
144         case 15: Keyboard('+', lastX, lastY);     break;
145         case 16: Keyboard('-', lastX, lastY);     break;
146         case 17: break;
147     }
148     glutPostRedisplay();
149 }
150
151 //---------------------------------------------------------------------------
152 void createMenu(void) {
153     int quadMenu = glutCreateMenu(menu);
154     glutAddMenuEntry("Add point             a", 3);
155     glutAddMenuEntry("Delete last point     d", 4);
156
157     glutAddMenuEntry("Increase red color    r", 5);
158     glutAddMenuEntry("Increase green color  g", 6);
159     glutAddMenuEntry("Increase blue color   b", 7);
160     glutAddMenuEntry("Increase alpha color  l", 8);
161
162     glutAddMenuEntry("Decrease red color    R", 9);
163     glutAddMenuEntry("Decrease green color  G", 10);
164     glutAddMenuEntry("Decrease blue color   B", 11);
165     glutAddMenuEntry("Decrease alpha color  L", 12);
166
167     glutAddMenuEntry("Rotate clockwise      Mouse wheel", 13);
168     glutAddMenuEntry("Rotate anticlockwise  Mouse wheel", 14);
169
170     glutAddMenuEntry("Enlarge               +", 15);
171     glutAddMenuEntry("Reduce                -", 16);
172
173     glutAddMenuEntry("Move                  Mouse", 17);
174
175     int mainMenu = glutCreateMenu(menu);
176     glutAddMenuEntry("Add Quad Strip     q", 1);
177     glutAddMenuEntry("Delete Quad Strip  Q", 2);
178     glutAddSubMenu("Recently used Quad Strip menu", quadMenu);
179
180     glutAttachMenu(GLUT_RIGHT_BUTTON);
181 }
182
183 //---------------------------------------------------------------------------
184 //---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void main(int argc, char *argv[]) {
    FreeConsole();

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Quad redactor");
    createMenu();

    // Устанавливаем функции, которые будут обрабатывать события
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Keyboard);
    glutMouseFunc(Mouse);
    glutMotionFunc(Motion);
    glutPassiveMotionFunc(PassiveMotion);
    glutMouseWheelFunc(MouseWheel);

    // Включаем сглаживание
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_BLEND);
    glEnable(GL_POINT_SMOOTH);
    glHint(GL_POINT_SMOOTH_HINT, GL_NICEST);
    glEnable(GL_LINE_SMOOTH);
    glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);

    // Создаем первый QuadStrip
    QuadStripInterface qq;
    qq.points.push_back(PointInterface(vec2(100, 100), 4));
    qq.points.push_back(PointInterface(vec2(200, 100), 4));
    qq.points.push_back(PointInterface(vec2(100, 200), 4));
    qq.points.push_back(PointInterface(vec2(200, 200), 4));
    qq.points.push_back(PointInterface(vec2(100, 300), 4));
    qq.points.push_back(PointInterface(vec2(200, 300), 4));
    quads.push_back(qq);

    glutMainLoop();
}
```

FILE **interface.h**

```
#pragma once

#include <vector>
#include <glm/glm.hpp>

using namespace glm;

//---------------------------------------------------------------------------
/** Функция вывода текста на экран средствами OpenGL. */
void printText(int x, int y, const char *string);

//---------------------------------------------------------------------------
/** Класс цвета. */
struct Color {
    Color() : r(255), g(255), b(255), a(255) {}
    Color(uint8_t r, uint8_t g, uint8_t b, uint8_t a) : r(r), g(g), b(b), a(a) {}
```

```
17
18      uint8_t r, g, b, a;
19  };
20
21  //---------------------------------------------------------------------------
22  /**
23      Класс передвигаемого объекта. Все объекты, наследуемые от него, могут быть
    ↪   перемещены на экране при помощи мыши.
24  */
25  class MovableObject {
26  public:
27      MovableObject() : m_isMoved(false), m_isActive(false), m_lastPos() {}
28
29      // Эти функции должен реализовать наследующий класс
30      virtual void display(void) = 0;
31      virtual bool isPointInside(vec2 point) const = 0;
32      virtual void offset(vec2 point) = 0;
33
34      // Эти функции реализованы текущим абстрактным классом
35      virtual bool processMouse(int button, int state, vec2 pos);
36      virtual bool processMotion(vec2 pos);
37      virtual bool processPassiveMotion(vec2 pos);
38
39      // Проверки насчет текущего состояния передвигаемого объекта
40      bool isActive(void) const;
41      bool isMoved(void) const;
42  private:
43      bool m_isMoved;
44      bool m_isActive;
45      vec2 m_lastPos;
46  };
47
48  //---------------------------------------------------------------------------
49  /** Класс перемещаемой с помощью мыши точки.
50      Во время неактивного состояния рисуется серым цветом.
51      Во время наведения мыши, рисуется оранжевым цветом.
52      Во время передвижения рисуется синим цветом.
53  */
54  class PointInterface : public MovableObject {
55  public:
56      PointInterface(vec2 pos, double r) : pos(pos), m_r(r) {}
57
58      void display(void);
59      bool isPointInside(vec2 point) const;
60      void offset(vec2 point);
61
62      vec2 pos;
63  private:
64      double m_r;
65  };
66
67  //---------------------------------------------------------------------------
68  /** Класс изменяемого с помощью мыши объекта Quad Strip. */
69  class QuadStripInterface : public MovableObject  {
70  public:
71      QuadStripInterface() : color(0, 0, 0, 65), m_drawColorInf(false) {}
72
73      void display(void);
74      bool isPointInside(vec2 point) const;
```

```
    void offset(vec2 point);

    bool processMouse(int button, int state, vec2 pos);
    bool processMotion(vec2 pos);
    bool processPassiveMotion(vec2 pos);
    bool processKeyboard(unsigned char key, vec2 pos);
    bool processWheel(int button, int dir, vec2 pos);

    Color color;
    std::vector<PointInterface> points;
private:
    bool m_drawColorInf;
};
```

FILE **interface.cpp**

```
#define GLM_ENABLE_EXPERIMENTAL

#include <GL/freeglut.h>
#include <string>
#include <glm/glm.hpp>
#include <glm/gtx/matrix_transform_2d.hpp>

#include ”interface.h”

//--------------------------------------------------------------------------
void printText(int x, int y, const char *string) {
    int len, i;

    glRasterPos2f(x, y);
    len = (int) strlen(string);
    for (i = 0; i < len; i++) {
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, string[i]);
    }
}


//--------------------------------------------------------------------------
bool MovableObject::processMouse(int button, int state, vec2 pos) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && isPointInside(pos)) {
        m_isMoved = true;
        m_lastPos = pos;
        return true;
    }

    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP && m_isMoved) {
        offset(pos - m_lastPos);
        m_isMoved = false;
        glutPostRedisplay();
        return true;
    }
    return false;
}

//--------------------------------------------------------------------------
bool MovableObject::processMotion(vec2 pos) {
    if (m_isMoved) {
        offset(pos - m_lastPos);
        m_lastPos = pos;
        glutPostRedisplay();
```

```
        return true;
    }
    return false;
}

//---------------------------------------------------------------------------
bool MovableObject::processPassiveMotion(vec2 pos) {
    if (isPointInside(pos)) {
        m_isActive = true;
        return true;
    } else {
        m_isActive = false;
        return false;
    }
}

//---------------------------------------------------------------------------
bool MovableObject::isActive(void) const {
    return m_isActive;
}

//---------------------------------------------------------------------------
bool MovableObject::isMoved(void) const {
    return m_isMoved;
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void PointInterface::display(void) {
    if (isMoved()) {
        glPointSize(2*m_r);
        glColor3ub(128, 128, 255);
    } else if (isActive()) {
        glPointSize(2*(m_r+1));
        glColor3ub(255, 128, 128);
    } else {
        glPointSize(2*m_r);
        glColor3ub(128, 128, 128);
    }
    glBegin(GL_POINTS);
    glVertex2f(pos.x, pos.y);
    glEnd();
}

//---------------------------------------------------------------------------
bool PointInterface::isPointInside(vec2 point) const {
    return length(point - pos) < m_r;
}

//---------------------------------------------------------------------------
void PointInterface::offset(vec2 point) {
    pos += point;
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
```

```
104 //---------------------------------------------------------------------------
105
106 //---------------------------------------------------------------------------
107 void QuadStripInterface::display(void) {
108     // Рисуем сам объект
109     glColor4ub(color.r, color.g, color.b, color.a);
110     glBegin(GL_QUAD_STRIP);
111     for (auto& i : points)
112         glVertex2f(i.pos.x, i.pos.y);
113     glEnd();
114
115     // Если он активен, то рисуем линии и точки на его вершинах
116     if (isActive()) {
117         glColor3ub(0, 0, 0);
118         for (int i = 0; i+4 <= points.size(); i += 2) {
119             vec2 a = points[i].pos, b = points[i+1].pos, c = points[i+3].pos, d
                ↪  = points[i+2].pos;
120
121             glBegin(GL_LINE_STRIP);
122             glVertex2f(a.x, a.y);
123             glVertex2f(b.x, b.y);
124             glVertex2f(c.x, c.y);
125             glVertex2f(d.x, d.y);
126             glVertex2f(a.x, a.y);
127             glEnd();
128         }
129
130         for (auto& i : points)
131             i.display();
132     }
133
134     // Выводим информацию о цвете, если изменяется цвет
135     if (m_drawColorInf) {
136         glColor3ub(0, 0, 0);
137         std::string r = "r: " + std::to_string(int(color.r));
138         std::string g = "g: " + std::to_string(int(color.g));
139         std::string b = "b: " + std::to_string(int(color.b));
140         std::string a = "a: " + std::to_string(int(color.a));
141         int offset = 15;
142         printText(5, 5+offset * 2, r.c_str());
143         printText(5, 5+offset * 1, g.c_str());
144         printText(5, 5+offset * 0, b.c_str());
145         printText(5, 5+offset * 3, a.c_str());
146         m_drawColorInf = false;
147     }
148 }
149
150 //---------------------------------------------------------------------------
151 bool QuadStripInterface::isPointInside(vec2 point) const {
152     auto isPointInsideTriangle = [] (vec2 v1, vec2 v2, vec2 v3, vec2 pt) -> bool
        ↪  {
153         auto sign = [] (vec2 p1, vec2 p2, vec2 p3) -> float {
154             return (p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x) * (p1.y - p3.y);
155         };
156
157         float d1, d2, d3;
158         bool has_neg, has_pos;
159
160         d1 = sign(pt, v1, v2);
```

```
        d2 = sign(pt, v2, v3);
        d3 = sign(pt, v3, v1);

        has_neg = (d1 < 0) || (d2 < 0) || (d3 < 0);
        has_pos = (d1 > 0) || (d2 > 0) || (d3 > 0);

        return !(has_neg && has_pos);
    };
    auto isPointInsideQuad = [&isPointInsideTriangle] (vec2 a, vec2 b, vec2 c, vec2 d, vec2 point) -> bool {
        return isPointInsideTriangle(a, b, c, point) || isPointInsideTriangle(a, c, d, point);
    };
    for (int i = 0; i+4 <= points.size(); i += 2) {
        if (isPointInsideQuad(points[i].pos, points[i+1].pos, points[i+3].pos, points[i+2].pos, point)) {
            return true;
        }
    }

    return false;
}

//-----------------------------------------------------------------------------
void QuadStripInterface::offset(vec2 point) {
    for (auto& i : points)
        i.pos += point;
}

//-----------------------------------------------------------------------------
bool QuadStripInterface::processMouse(int button, int state, vec2 pos) {
    // Сначала обрабатываются точки, если объект не перемещается, а лишь затем обрабатывается сам объект
    if (isActive() && !isMoved()) {
        for (auto& i : points) {
            if (i.processMouse(button, state, pos))
                return true;
        }
    }

    return MovableObject::processMouse(button, state, pos);
}

//-----------------------------------------------------------------------------
bool QuadStripInterface::processMotion(vec2 pos) {
    // Сначала обрабатываются точки, если объект не перемещается, а лишь затем обрабатывается сам объект
    if (isActive() && !isMoved()) {
        for (auto& i : points) {
            if (i.processMotion(pos))
                return true;
        }
    }

    return MovableObject::processMotion(pos);
}

//-----------------------------------------------------------------------------
bool QuadStripInterface::processPassiveMotion(vec2 pos) {
```

```
    // Сначала обрабатываются точки, если объект не перемещается, а лишь затем обрабатывается сам объект
    if (isActive() && !isMoved()) {
        for (auto& i : points) {
            if (i.processPassiveMotion(pos))
                return true;
        }
    }

    return MovableObject::processPassiveMotion(pos);
}

//------------------------------------------------------------------------------
bool QuadStripInterface::processKeyboard(unsigned char key, vec2 pos) {
    if (isActive()) {
        // Здесь обрабатывается только увеличение в размерах и изменение цвета
        if (key == '=' || key == '+') {
            for (auto& i : points) {
                i.pos -= pos;
                i.pos *= 1.2;
                i.pos += pos;
            }
        }
        if (key == '-') {
            for (auto& i : points) {
                i.pos -= pos;
                i.pos /= 1.2;
                i.pos += pos;
            }
        }

        if (key == 'r') { color.r += 5; m_drawColorInf = true; }
        if (key == 'g') { color.g += 5; m_drawColorInf = true; }
        if (key == 'b') { color.b += 5; m_drawColorInf = true; }
        if (key == 'l') { color.a += 5; m_drawColorInf = true; }

        if (key == 'R') { color.r -= 5; m_drawColorInf = true; }
        if (key == 'G') { color.g -= 5; m_drawColorInf = true; }
        if (key == 'B') { color.b -= 5; m_drawColorInf = true; }
        if (key == 'L') { color.a -= 5; m_drawColorInf = true; }

        return true;
    } else
        return false;
}

//------------------------------------------------------------------------------
bool QuadStripInterface::processWheel(int button, int dir, vec2 pos) {
    // Обработка мыши для вращения всего объекта вокруг мыши
    mat3 rotate1 = translate(rotate(translate(glm::mat3(1.0f), pos), radians(3.0f)), -pos);
    mat3 rotate2 = translate(rotate(translate(glm::mat3(1.0f), pos), radians(-3.0f)), -pos);

    if (isActive()) {
        if (dir > 0) {
            for (auto& i : points) {
                auto res = rotate1 * vec3(i.pos.x, i.pos.y, 1);
                i.pos = vec2(res.x, res.y);
```

```
            }
        } else {
            for (auto& i : points) {
                auto res = rotate2 * vec3(i.pos.x, i.pos.y, 1);
                i.pos = vec2(res.x, res.y);
            }
        }
        return true;
    } else
        return false;
}
```