|  |  |  |
| --- | --- | --- |
| Министерство образования и науки Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
| path817.png | | |
| Кафедра прикладной математики | | |
|  | | |
| Лабораторная работа № 2 | | |
| по дисциплине «Компьютерная графика» | | |
|  | | |
|  | | |
|  | | |
| эмблема_светлая.png | Факультет: | ПМИ |
| Группа: | ПМИ-62 |
| Бригада: | 4 |
| Студенты: | Мамонова Е. В., Ершов П. К. |
| Преподаватель: | Задорожный А. Г. |
|  | | |
| Новосибирск | | |
| 2019 | | |

1. Цель работы

Ознакомиться с принципами растеризации примитивов и наложения цветов и текстур.

1. **Задание**
   1. В соответствии с заданием отобразить объекты (правильные многоугольники), центр и «радиус» которых определяются кликами мыши.
   2. Предоставить возможность изменять форму и положение объектов модельно-видовыми преобразованиями.
   3. Предоставить возможность изменять цвет и текстуру текущего (активного) объекта.
   4. Предоставить возможность включать и выключать логические операции смешивания цветов (в соответствии с вариантом задания).
   5. Вывести на экране динамически меняемую крупную сетку условных пикселей и на ней растеризовать данные объекты в режимах «контур» и «с заполнением».
   6. При растеризации пересекающихся объектов цвет области пересечения определять в соответствии с заданной операцией смешения цветов.
   7. Продублировать команды в меню, созданном с помощью библиотеки GLUT.
2. **Описание управления**

Меню включает в себя:

Color – меню цветов.

Scale- меню размера.

Texture – меню текстур.

Logic operations –меню логических операций с цветом.

Rasterization – меню растеризации.

Управление клавишами.

Space – включить/выключить режим редактирования точек.

Delet – удалить многоугольник.

W/w – переместить многоугольник вверх.

A/a – переместить многоугольник

S/s – переместить многоугольник вниз.

D/d – переместить многоугольник вправо.

L/l – сменить режим логической операции с цветами.

P/p – удалить все фигуры.

+/- – увеличить/ уменьшить радиус фигуры.

N/n – сменить режим растеризации.

M/m – изменить размер сетки растеризации.

1. Текст программы

pch.h

#ifndef PCH\_H

#define PCH\_H

#define GLUT\_DISABLE\_ATEXIT\_HACK

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <windows.h>

#include "glut.h"

#include <vector>

#include <GL\GL.h>

#include <GL\GLU.h>

#include <algorithm>

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

#pragma comment (lib, "opengl32.lib")

#pragma comment (lib, "glu32.lib")

#pragma comment(lib,"glut32.lib")

using namespace std;

GLint width, height; //ширина и высота окна

int current\_figure; //текущая фигура

int max\_figure; //общее число фигур

int N = 8;

bool drawing\_mode; //true - режим отрисовки

bool dot\_mode; //режим редактирования точек

int current\_dot; //текущая точка

int main\_menu1; //идентификатор меню работы с шестиугольниками

int main\_menu2; //идентификатор меню работы с точками

unsigned int names\_tex[8]; //имена структур

int colour\_shuffle\_mode; //режим смешения цветов: 0 - нет, 1 - and, 2 - not and

int netsize; //шаг сетки

vector <pair<int, int>> net\_line;

class point

{

public:

GLint x, y;

point(GLint p1, GLint p2)

{

x = p1;

y = p2;

}

};

class pointf //вещественная точка/использовалась для расчета текстурных координат/

{

public:

GLfloat x, y;

pointf(GLfloat p1, GLfloat p2)

{

x = p1;

y = p2;

}

};

class colour

{

public:

GLint red, blue, green;

colour(GLint r, GLint b, GLint g)

{

red = r;

blue = b;

green = g;

}

};

struct active\_edge

{

double k;

int x, y, xk, yk;

int dx, dy;

int yfin;

}a;

class polygon

{

public:

vector <point> coord\_polygon; //координаты восьмиугольника

vector <colour> color\_point; //цвет координаты восьмиугольника

int Ox; //сдвиг по оси Ох от начала координат

int Oy; //сдвиг по оси Оу от начала координат

GLint center\_x;

GLint center\_y;

//координаты первой вершины восьмиугольника

GLint vertex\_x;

GLint vertex\_y;

double ScaleOx; //коэффициент масштабирования по Ох

double ScaleOy; //коэффициент масштабирования по Оу

int Angle; //угол поворота

int number; //номер восьмиугольника

int texture\_number; //номер текстуры -1 если нет текстуры

int overlay\_type\_tex; //тип наложения текстуры

bool cent\_stat;//false - центр еще не установлен, или координаты фигуры определены окончательно

bool cord\_stat;//true - можно рассчитывать координаты восьмиугольника

};

extern bool group(const pair<int, int> &a, const pair <int, int> &b);

vector <polygon> v; //вектор восьмиугольников

polygon cen\_ver; //содержит координаты центра и первой вершины отрисовываемого восьмиугольника

vector <point> drawing\_v; //вектор для расчета координат отрисовываемого восьмиугольника

vector <pointf> texture\_coord; //текстурные координаты/считаются один раз и на века функцией calc\_texture\_coord/

polygon drawing\_polygon; //отрисовываемый восьмиугольник/нужен только для установки стандартных параметров отрисованного восьмиугольника, потом сразу скидывается в v и вычищается/

colour drawing\_color = colour(150, 150, 150); //цвет по умолчанию

int raster\_mode; //режим растеризации, 0 - нет, 1 - контур, 2 - заполнение

vector <int> raster\_size; //массив, своими значениями разбивающий массив сеткелей на принадлежность к фигурам

extern int rasterize(const polygon & n, vector <pair<int, int>> \* v); //алгоритм заполнения многоугольника

extern bool group(const pair<int, int> &a, const pair <int, int> &b);

void Change\_Mode();

void On\_Up\_Key();

void On\_Down\_Key();

void MoveUp();

void MoveDown();

void MoveLeft();

void MoveRight();

void Increase(double zoom);

void Reduce(double zoom);

void Del\_Group();//удалить фигуру

void Clear();//удалить все фигуры

void RotateLeft();

void RotateRight();

bool equality(point one, point two);

void Finish\_Primitive();;

void calc\_polygon\_coord(point center, point vertex);//вычисление координат шестиугольника

void set\_texture\_coord();//расчет текстурных координат

void processColorMenu(int option);//обработчик подменю цветов mainmenu1

void processColorMenu2(int option);//обработчик подменю цветов mainmenu2

void processScaleMenu(int option);

void processTextureMenuMain(int option);

void processTextureMenu(int option);

void processOverlayMenu(int option);

void processLogicMenu(int option);

void processRasterMenu(int option);

void processRasterModeMenu(int option);

void processNetMenu(int option);

void processMainMenu(int option);

void createMenu();

void loadTexture();

void SwitchColourShuffleMode();

void SwitchRasterMode();

void SwitchNetSize();

bool point\_in\_net(int x0, int y0, int \* xi, int \* yj); //координаты искомой точки, координаты сеткеля

void quater\_net(int x, int y, int i, int j, int \* xn, int \* yn);

int contour\_Bresenham(int x\_0, int y\_0, int x\_k, int y\_k, vector <pair<int, int>> \* v, int cx, int cy);

bool pred(const pair<int, int> &a, const pair <int, int> &b);

//сортировка по возрастанию внутри группы

bool group(const pair<int, int> &a, const pair <int, int> &b);

void Process\_Mouse\_Move(int x, int y);

void Process\_Mouse(int button, int state, int x, int y);

void Display(void);

void Process\_Normal\_Keys(unsigned char key, int x, int y);

void Process\_Special\_Keys(int key, int x, int y);

void Initialize();

void Reshape(GLint w, GLint h);

#endif //PCH\_H

pch.cpp

#include "pch.h"

void Change\_Mode()//включение выключение режима редактирования точек

{

char str[50];

if (dot\_mode)//если режим редактирования точек был включен

{

sprintf(str, "dot mod OFF");

glutSetWindowTitle(str);

dot\_mode = false; //выключить режим редактирования точек

glutSetMenu(main\_menu1); //изменить идентификатор меню

glutAttachMenu(GLUT\_RIGHT\_BUTTON);//назначить меню на ПКМ

}

else

{

current\_dot = 0;

sprintf(str, "dot mod ON");

glutSetWindowTitle(str);

dot\_mode = true; //включить режим редактирования точек

glutSetMenu(main\_menu2); //изменить идентификатор меню

glutAttachMenu(GLUT\_RIGHT\_BUTTON);//назначить меню на ПКМ

}

}

void On\_Up\_Key()//передвижение по всем шестиугольникам, либо по всем точкам в режиме редактирования точек

{

char str[50];

if (!dot\_mode) //если не режим редактирования точек

{

//перейти к следующему шестиугольнику

current\_figure++;

//перейти к нулевому, если был достигнут последний добавленный шестиугольник

current\_figure = current\_figure % (max\_figure);

sprintf(str, "Current figure: %i", current\_figure + 1);

glutSetWindowTitle(str);

}

else

{

current\_dot++; //перейти к следующей точке

current\_dot = current\_dot % N; //перейти к нулевой точке, если была достигнута последняя точка в шестиугольнике

sprintf(str, "Current dot: %i", current\_dot + 1);

glutSetWindowTitle(str);

}

}

void On\_Down\_Key()//передвижение по всем шестиугольникам, либо по всем точкам в режиме редактирования точек

{

char str[50];

if (!dot\_mode)

{

//перейти к предыдущему шестиугольнику

current\_figure--;

//перейти к последнему добавленному шестиугольнику, если был пройден нулевой шестиугольник

if (current\_figure < 0) current\_figure = max\_figure - 1;

sprintf(str, "Current figure: %i", current\_figure + 1);

glutSetWindowTitle(str);

}

else

{

current\_dot--; //перейти к предыдущей точке

if (current\_dot < 0) current\_dot = 5; //перейти к последней точке, если была достигнута нулевая точка в шестиугольнике

sprintf(str, "Current dot: %i", current\_dot + 1);

glutSetWindowTitle(str);

}

}

void Del\_Group()

{

char str[50];

v.erase(v.begin() + current\_figure);

max\_figure--;

sprintf(str, "%i groups", max\_figure + 1);

glutSetWindowTitle(str);

}

void Clear()

{

char str[50];

v.clear();

max\_figure = 0;

sprintf(str, "%i groups", max\_figure + 1);

glutSetWindowTitle(str);

}

void MoveUp()

{

if (v[current\_figure].Oy + v[current\_figure].ScaleOy <= height)

v[current\_figure].Oy += 10;

}

void MoveDown()

{

if (v[current\_figure].Oy - v[current\_figure].ScaleOy >= 0)

v[current\_figure].Oy -= 10;

}

void MoveLeft()

{

if (v[current\_figure].Ox - v[current\_figure].ScaleOx >= 0)

v[current\_figure].Ox -= 10;

}

void MoveRight()

{

if (v[current\_figure].Ox + v[current\_figure].ScaleOx <= width)

v[current\_figure].Ox += 10;

}

void Increase(double zoom)

{

v[current\_figure].ScaleOx += zoom;

v[current\_figure].ScaleOy += zoom;

}

void Reduce(double zoom)

{

if (v[current\_figure].ScaleOx - zoom > 0 && v[current\_figure].ScaleOy - zoom > 0)

{

v[current\_figure].ScaleOx -= zoom;

v[current\_figure].ScaleOy -= zoom;

}

}

void RotateLeft()

{

v[current\_figure].Angle += 5;

if (v[current\_figure].Angle > 360)

v[current\_figure].Angle %= 360;

}

void RotateRight()

{

v[current\_figure].Angle -= 5;

if (v[current\_figure].Angle < 360)

v[current\_figure].Angle %= 360;

}

bool equality(point one, point two)//проверка равенства двух точек

{

if (one.x == two.x && one.y == two.y)

return true;

return false;

}

void Finish\_Primitive()

{

char str[50];

//отключить режим рисования

drawing\_mode = false;

//если координаты центра и вершины не совпадают

if (!equality(point(cen\_ver.center\_x, cen\_ver.center\_y), point(cen\_ver.vertex\_x, cen\_ver.vertex\_y)))

{

current\_figure = max\_figure;

//установка атрибутов фигуры

drawing\_polygon.number = current\_figure;

//сдвиг фигуры от начала координат равен центру фигуры

drawing\_polygon.Ox = cen\_ver.center\_x;

drawing\_polygon.Oy = cen\_ver.center\_y;

//масштаб = не масштабирована

drawing\_polygon.ScaleOx = 1;

drawing\_polygon.ScaleOy = 1;

//угол поворота = 0

drawing\_polygon.Angle = 0;

//текстура отсутствует

drawing\_polygon.texture\_number = -1;

//тип наложения текстуры

drawing\_polygon.overlay\_type\_tex = 1;

for (int i = 0; i < drawing\_v.size(); i++)

{

//поместить текущий цвет вершин в отрисовываемый шестиугольник

drawing\_polygon.color\_point.push\_back(drawing\_color);

//поместить текущие вершины в отрисовываемый шестиугольник

drawing\_polygon.coord\_polygon.push\_back(drawing\_v[i]);

}

v.push\_back(drawing\_polygon);//поместить отрисовываемый шестиугольник в вектор

//очистка векторов

drawing\_v.erase(drawing\_v.begin(), drawing\_v.end());

drawing\_polygon.color\_point.erase(drawing\_polygon.color\_point.begin(), drawing\_polygon.color\_point.end());

drawing\_polygon.coord\_polygon.erase(drawing\_polygon.coord\_polygon.begin(), drawing\_polygon.coord\_polygon.end());

sprintf(str, "Current figure: %i", current\_figure + 1);

glutSetWindowTitle(str);

max\_figure++;

}

else //если ткнули два раза в одну точку, добавлять нечего

current\_figure = max\_figure - 1; //вернуться к предыдущему шестиугольнику

}

void calc\_polygon\_coord(point center, point vertex)//вычисление координат шестиугольника

{

int n = N; //число вершин

double radius; //радиус описанной окружности

double A0; //угловая координата первой вершины

double centr\_x = double(center.x);

double centr\_y = double(center.y);

double vert\_x = double(vertex.x);

double vert\_y = double(vertex.y);

double cur\_x, cur\_y;

drawing\_v.erase(drawing\_v.begin(), drawing\_v.end()); //очистить вектор

A0 = atan((vert\_x - centr\_x) / (vert\_y - centr\_y));

if (vert\_y < centr\_y)

A0 += M\_PI;

radius = sqrt(pow((vert\_x - centr\_x), 2.0) + pow((vert\_y - centr\_y), 2.0));

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

{

//координаты рассчитываются от начала координатной оси (0,0)

cur\_x = radius \* sin(A0 + 2 \* M\_PI\*i / n);

cur\_y = radius \* cos(A0 + 2 \* M\_PI\*i / n);

point p = point(cur\_x, cur\_y);

drawing\_v.push\_back(p);

}

}

void set\_texture\_coord()//расчет текстурных координат

{

double A = M\_PI / N;

pointf coord = pointf(0.5, 1);

texture\_coord.push\_back(coord);

coord = pointf(0.5 + 0.5\*cos(A), 0.75);

texture\_coord.push\_back(coord);

coord = pointf(0.5 + 0.5\*cos(A), 0.25);

texture\_coord.push\_back(coord);

coord = pointf(0.5, 0);

texture\_coord.push\_back(coord);

coord = pointf(0.5 - 0.5\*cos(A), 0.25);

texture\_coord.push\_back(coord);

coord = pointf(0.5 - 0.5\*cos(A), 0.75);

texture\_coord.push\_back(coord);

coord = pointf(0.5, 1);

texture\_coord.push\_back(coord);

coord = pointf(0.5 - 0.5\*cos(A), 0.75);

texture\_coord.push\_back(coord);

}

void processColorMenu(int option)//обработчик подменю цветов mainmenu1

{

if (!drawing\_mode && max\_figure != 0)

{

colour color = colour(0, 0, 0);

switch (option)

{

case 1:

color.red = 255;

color.green = 0;

color.blue = 0;

for (int i = 0; i < v[current\_figure].coord\_polygon.size(); i++)

v[current\_figure].color\_point[i] = color;

break;

case 3:

color.red = 10;

color.green = 240;

color.blue = 10;

for (int i = 0; i < v[current\_figure].coord\_polygon.size(); i++)

v[current\_figure].color\_point[i] = color;

break;

case 2:

color.red = 0;

color.green = 0;

color.blue = 255;

for (int i = 0; i < v[current\_figure].coord\_polygon.size(); i++)

v[current\_figure].color\_point[i] = color;

break;

case 4:

color.red = 246;

color.green = 138;

color.blue = 42;

for (int i = 0; i < v[current\_figure].coord\_polygon.size(); i++)

v[current\_figure].color\_point[i] = color;

break;

case 5:

color.red = 246;

color.green = 224;

color.blue = 42;

for (int i = 0; i < v[current\_figure].coord\_polygon.size(); i++)

v[current\_figure].color\_point[i] = color;

break;

case 6:

color.red = 182;

color.green = 97;

color.blue = 183;

for (int i = 0; i < v[current\_figure].coord\_polygon.size(); i++)

v[current\_figure].color\_point[i] = color;

break;

case 7:

color.red = 0;

color.green = 0;

color.blue = 0;

for (int i = 0; i < v[current\_figure].coord\_polygon.size(); i++)

v[current\_figure].color\_point[i] = color;

break;

case 8:

color.red = 255;

color.green = 255;

color.blue = 255;

for (int i = 0; i < v[current\_figure].coord\_polygon.size(); i++)

v[current\_figure].color\_point[i] = color;

break;

}

glutPostRedisplay();

}

}

void processColorMenu2(int option)//обработчик подменю цветов mainmenu2

{

if (!drawing\_mode && max\_figure != 0)

{

colour color = colour(0, 0, 0);

switch (option)

{

case 1:

color.red = 252;

color.green = 48;

color.blue = 58;

v[current\_figure].color\_point[current\_dot] = color;

break;

case 3:

color.red = 105;

color.green = 197;

color.blue = 22;

v[current\_figure].color\_point[current\_dot] = color;

break;

case 2:

color.red = 42;

color.green = 128;

color.blue = 245;

v[current\_figure].color\_point[current\_dot] = color;

break;

case 4:

color.red = 246;

color.green = 138;

color.blue = 42;

v[current\_figure].color\_point[current\_dot] = color;

break;

case 5:

color.red = 246;

color.green = 224;

color.blue = 42;

v[current\_figure].color\_point[current\_dot] = color;

break;

case 6:

color.red = 182;

color.green = 97;

color.blue = 183;

v[current\_figure].color\_point[current\_dot] = color;

break;

case 7:

color.red = 0;

color.green = 0;

color.blue = 0;

v[current\_figure].color\_point[current\_dot] = color;

break;

case 8:

color.red = 255;

color.green = 255;

color.blue = 255;

v[current\_figure].color\_point[current\_dot] = color;

break;

}

glutPostRedisplay();

}

}

void processScaleMenu(int option)

{

if (!drawing\_mode && max\_figure != 0)

{

switch (option)

{

case 1:

Increase(0.3);

break;

case 2:

Reduce(0.3);

break;

}

glutPostRedisplay();

}

}

void processTextureMenuMain(int option)

{

}

void processTextureMenu(int option)

{

if (!drawing\_mode && max\_figure != 0)

{

switch (option)

{

case 1:

v[current\_figure].texture\_number = 0;

break;

case 2:

v[current\_figure].texture\_number = 1;

break;

case 3:

v[current\_figure].texture\_number = 2;

break;

case 4:

v[current\_figure].texture\_number = 3;

break;

case 5:

v[current\_figure].texture\_number = 4;

break;

case 6:

v[current\_figure].texture\_number = 5;

break;

case 7:

v[current\_figure].texture\_number = 6;

break;

case 8:

v[current\_figure].texture\_number = 7;

break;

case 9:

v[current\_figure].texture\_number = -1;

v[current\_figure].overlay\_type\_tex = 1;

break;

}

glutPostRedisplay();

}

}

void processOverlayMenu(int option)

{

if (!drawing\_mode && max\_figure != 0)

{

switch (option)

{

case 1:

v[current\_figure].overlay\_type\_tex = 1;

break;

case 2:

v[current\_figure].overlay\_type\_tex = 2;

break;

}

glutPostRedisplay();

}

}

void processLogicMenu(int option)

{

char str[50];

if (!drawing\_mode && !dot\_mode && max\_figure != 0)

{

colour\_shuffle\_mode = option;

glutPostRedisplay();

sprintf(str, "Current logic operation: %s", colour\_shuffle\_mode ? (colour\_shuffle\_mode == 1 ? "AND" : "NAND") : "none");

glutSetWindowTitle(str);

}

}

void processRasterMenu(int option)

{

}

void processRasterModeMenu(int option)

{

char str[50];

if (!drawing\_mode && !dot\_mode)

{

raster\_mode = option;

glutPostRedisplay();

sprintf(str, "Current rasterization: %s", raster\_mode ? (raster\_mode == 1 ? "contour" : "fill") : "turned off");

glutSetWindowTitle(str);

}

}

void processNetMenu(int option)

{

char str[50];

if (!drawing\_mode && !dot\_mode && raster\_mode)

{

netsize = option;

glutPostRedisplay();

sprintf(str, "Current net size: %i", netsize);

glutSetWindowTitle(str);

}

}

void processMainMenu(int option)

{

};

void createMenu()

{

int color\_menu1;

int color\_menu2;

int scale\_menu;

int texture\_menu\_main;

int texture\_menu;

int overlay\_menu;

int logic\_menu;

int raster\_menu;

int raster\_mode\_menu;

int raster\_size\_menu;

//подменю цвет

color\_menu1 = glutCreateMenu(processColorMenu);

glutAddMenuEntry("Red", 1);//добавить пункты подменю

glutAddMenuEntry("Blue", 2);

glutAddMenuEntry("Green", 3);

glutAddMenuEntry("Orange", 4);

glutAddMenuEntry("Yellow", 5);

glutAddMenuEntry("Violet", 6);

glutAddMenuEntry("Black", 7);

glutAddMenuEntry("White", 8);

//подменю цвет для работы с точками

color\_menu2 = glutCreateMenu(processColorMenu2);

glutAddMenuEntry("Red", 1);//добавить пункты подменю

glutAddMenuEntry("Blue", 2);

glutAddMenuEntry("Green", 3);

glutAddMenuEntry("Orange", 4);

glutAddMenuEntry("Yellow", 5);

glutAddMenuEntry("Violet", 6);

glutAddMenuEntry("Black", 7);

glutAddMenuEntry("White", 8);

//подменю масштаб

scale\_menu = glutCreateMenu(processScaleMenu);

glutAddMenuEntry("Zoom in", 1);//добавить пункты подменю

glutAddMenuEntry("Zoom out", 2);

//подподменю типа текстуры

texture\_menu = glutCreateMenu(processTextureMenu);

glutAddMenuEntry("Texture 1", 1);//добавить пункты подменю

glutAddMenuEntry("Texture 2", 2);

glutAddMenuEntry("Texture 3", 3);

glutAddMenuEntry("Texture 4", 4);

glutAddMenuEntry("Texture 5", 5);

glutAddMenuEntry("Texture 6", 6);

glutAddMenuEntry("Texture 7", 7);

glutAddMenuEntry("Texture 8", 8);

glutAddMenuEntry("Delete texture", 9);

//подподменю типа наложения текстуры

overlay\_menu = glutCreateMenu(processOverlayMenu);

glutAddMenuEntry("Multiplication", 1);//добавить пункты подменю

glutAddMenuEntry("Firm texture", 2);

//подменю текстуры

texture\_menu\_main = glutCreateMenu(processTextureMenuMain);

glutAddSubMenu("Type texture", texture\_menu);

glutAddSubMenu("Type overlay", overlay\_menu);

//главное меню в режиме редактирования точек

main\_menu2 = glutCreateMenu(processMainMenu);

glutAddSubMenu("Color", color\_menu2);

//подменю логических операций

logic\_menu = glutCreateMenu(processLogicMenu);

glutAddMenuEntry("none", 0);

glutAddMenuEntry("AND", 1);

glutAddMenuEntry("NAND", 2);

//подподменю типа режима растеризации

raster\_mode\_menu = glutCreateMenu(processRasterModeMenu);

glutAddMenuEntry("Turn off", 0);

glutAddMenuEntry("Contour", 1);

glutAddMenuEntry("Fill", 2);

//подподменю типа размера сетки

raster\_size\_menu = glutCreateMenu(processNetMenu);

glutAddMenuEntry("6", 6);

glutAddMenuEntry("10", 10);

glutAddMenuEntry("14", 14);

glutAddMenuEntry("18", 18);

glutAddMenuEntry("20", 22);

//подменю растеризации

raster\_menu = glutCreateMenu(processRasterMenu);

glutAddSubMenu("Rasterization mode", raster\_mode\_menu);

glutAddSubMenu("Net size", raster\_size\_menu);

//главное меню в режиме редактирования шестиугольников

main\_menu1 = glutCreateMenu(processMainMenu);

glutAddSubMenu("Color", color\_menu1);

glutAddSubMenu("Scale", scale\_menu);

glutAddSubMenu("Texture", texture\_menu\_main);

glutAddSubMenu("Logic operations", logic\_menu);

glutAddSubMenu("Rasterization", raster\_menu);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);//прикрепить меню к правой кнопке мыши

}

void loadTexture()

{

FILE \*F;

char str[20] = "1.bmp";

char num = '1';

auto widthT = width;

auto heightT = height;

glGenTextures(8, names\_tex); //создание имен текстур

for (int i = 0; i < 8; i++)

{

str[0] = num;

num++;

if ((F = fopen(str, "rb")) != NULL)

{

/\*Перемещаемся в bmp-файле на нужную позицию, и считываем ширину и длинну \*/

fseek(F, 18, SEEK\_SET);

fread(&(widthT), 2, 1, F);

fseek(F, 2, SEEK\_CUR);

fread(&(heightT), 2, 1, F);

/\* Выделяем память под изображение. Если память не выделилась, закрываем файл и выходим с ошибкой \*/

auto pixels = new GLubyte[3 \* widthT \* heightT];

/\* Считываем изображение в память по 3 бита, то бишь RGB для каждого пикселя \*/

fseek(F, 30, SEEK\_CUR);

fread(pixels, 3, widthT \* heightT, F);

glBindTexture(GL\_TEXTURE\_2D, names\_tex[i]); //выбрать текущую текстуру

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

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

gluBuild2DMipmaps(GL\_TEXTURE\_2D, 3, widthT, heightT, GL\_BGR\_EXT, GL\_UNSIGNED\_BYTE, pixels);

free(pixels);

fclose(F);

}

}

}

void SwitchColourShuffleMode()

{

char str[50];

colour\_shuffle\_mode = (colour\_shuffle\_mode + 1) % 3;

sprintf(str, "Current logic operation: %s", colour\_shuffle\_mode ? (colour\_shuffle\_mode == 1 ? "AND" : "NAND") : "none");

glutSetWindowTitle(str);

}

void SwitchRasterMode()

{

char str[50];

raster\_mode = (raster\_mode + 1) % 4;

sprintf(str, "Current rasterization: %i", raster\_mode);

glutSetWindowTitle(str);

}

void SwitchNetSize()

{

char str[50];

netsize = netsize + 2;

if (netsize > 9) netsize = 4;

sprintf(str, "Current net size: %i", netsize);

glutSetWindowTitle(str);

}

//сеткель - элемент искусственной растровой сетки, для простоты дальнейших комментариев

bool point\_in\_net(int x0, int y0, int \* xi, int \* yj) //координаты искомой точки, координаты сеткеля

{

bool found = false;

int fal\_width, fal\_height;

int i, j;

fal\_width = 2.2 \* width; //точки шестиугольников могут уходить за видимые пределы окна

fal\_height = 2.2 \* height; //поэтому поиск точки происходит в значительно бо'льших пределах

for (i = -fal\_width / netsize - 1; i < fal\_width / netsize + 1 && !found; i++)

{

for (j = -fal\_height / netsize - 1; j < fal\_height / netsize + 1 && !found; j++)

{

if (x0 >= i \* netsize && x0 < (i + 1)\*netsize)

{

if (y0 >= j \* netsize && y0 < (j + 1)\*netsize)

{

found = true;

}

}

}

}

\*xi = i;

\*yj = j;

return (found);

}

void quater\_net(int x, int y, int i, int j, int \* xn, int \* yn)

{

switch (netsize)

{

case 4:

\*xn = i \* 4 - 2;

\*yn = j \* 4 - 2;

break;

case 6:

if (x > i \* 6 - 3)

\*xn = i \* 6 - 2;

else

if (x < i \* 6 - 3)

\*xn = i \* 6 - 4;

if (y > j \* 6 - 3)

\*yn = j \* 6 - 2;

else

if (y < j \* 6 - 3)

\*yn = j \* 6 - 4;

break;

case 8:

if (x > i \* 8 - 4)

\*xn = i \* 6 - 3;

else

\*xn = i \* 6 - 5;

if (y > j \* 8 - 4)

\*yn = j \* 6 - 3;

else

\*yn = j \* 6 - 5;

break;

case 10:

if (x > i \* 10 - 5)

\*xn = i \* 10 - 4;

else

\*xn = i \* 10 - 6;

if (y > j \* 10 - 5)

\*yn = j \* 10 - 4;

else

\*yn = j \* 10 - 6;

break;

}

}

int contour\_Bresenham(int x\_0, int y\_0, int x\_k, int y\_k, vector <pair<int, int>> \* v, int cx, int cy)

{

int octant, i, j;

int K;

bool found = false;

double coef\_angle; //угловой коэффициент

int temp;

int x0, y0, xk, yk;

x0 = x\_0;

y0 = y\_0;

xk = x\_k;

yk = y\_k;

coef\_angle = (double)(yk - y0) / (double)(xk - x0); //угловой коэффициент кривой

if (xk > x0) //расчет октанта, в котором расположен отрезок, от октанта зависят переменные

{

if (yk > y0)

{

if (abs(coef\_angle) > 1.0L)

{

octant = 8;

}

else

{

octant = 7;

}

}

else

{

if (abs(coef\_angle) > 1.0L)

{

octant = 5;

}

else

{

octant = 6;

}

}

}

else

{

if (yk > y0)

{

if (abs(coef\_angle) > 1.0L)

{

octant = 1;

}

else

{

octant = 2;

}

}

else

{

if (abs(coef\_angle) > 1.0L)

{

octant = 4;

}

else

{

octant = 3;

}

}

}

found = point\_in\_net(x0, y0, &i, &j);

int k = -1;

if (found) //определена принадлежность точки начала отрезка сеткелю [i, j]

{

double t;

int dx, dy, sy, sx;

dx = abs(xk - x0);

dy = abs(yk - y0);

if (xk > x0) sx = 1;

else sx = -1;

if (yk > y0) sy = 1;

else sy = -1;

if (dy > dx)//одно из направлений меняется на 1 на каждом шаге, поэтому кол-во сеткелей равно кол-ву отрезков ширины сеткеля, которое можно уложить в этом направлении (+1 потому что первая точка (вершина) кладется сразу, без расчетов)

{

K = dy / netsize + 1;

}

else

{

K = dx / netsize + 1;

}

int x, y;

int error2;

x = x0;

y = y0;

int fini, finj;

k = 0;

if (xk % netsize < 1)

{

if (xk < cx) xk += 1;

else xk -= 1;

}

if (yk % netsize < 1)

{

if (yk < cy) yk += 1;

else yk -= 1;

}

if (x0 % netsize < 1)

{

if (x0 < cx) x0 += 1;

else x0 -= 1;

}

if (y0 % netsize < 1)

{

if (y0 < cy) y0 += 1;

else y0 -= 1;

}

found = point\_in\_net(xk, yk, &fini, &finj);

found = point\_in\_net(x0, y0, &i, &j);

int x0i, y0j, xki, ykj;

x0i = x0;

y0j = y0;

xki = xk;

ykj = yk;

dx = abs(xki - x0i);

dy = abs(ykj - y0j);

int error = dx - dy;

while ((i != fini || j != finj) && k < K)

{

v->push\_back(make\_pair(i, j));

error2 = error \* 2;

if (error2 > -dy)

{

error -= dy;

i += sx;

}

if (error2 < dx)

{

error += dx;

j += sy;

}

k++;

}

found = point\_in\_net(xk, yk, &i, &j);

v->push\_back(make\_pair(i, j));

}

else return 0;

return (k + 1);

}

bool pred(const pair<int, int> &a, const pair <int, int> &b)

{

return a.second > b.second;

}

//сортировка по возрастанию внутри группы

bool group(const pair<int, int> &a, const pair <int, int> &b)

{

if (a.second != b.second) return a.second > b.second;

else

return a.first < b.first;

}

int rasterize(const polygon & n, vector <pair<int, int>> \* v)

{

vector <pair<int, int>> nodes; //узлы в пикселях

vector <pair<int, int>> w; //все скидывается сюда, в конце подпрограммы переписывается в \*v

vector <pair<int, int>> nodes\_net; //узлы в квадратах растной сетки

int i, j;

int x, y, ymin, ycur;

int temp;

int K; //число добавляемых квадратов растной сетки

for (i = 0; i < N; i++) //получили узлы фигуры

{

x = n.coord\_polygon[i].x + n.Ox;

y = n.coord\_polygon[i].y + n.Oy;

nodes.push\_back(make\_pair(x, y));

}

sort(nodes.begin(), nodes.end(), pred); //отсортировали по y

if (nodes[N - 1].second == nodes[N - 2].second && nodes[N - 1].first < nodes[N - 2].first) //если нижнее ребро горизонтально, надо убедится, что левый узел действительно левее правого

{

temp = nodes[N - 1].first;

nodes[N - 1].first = nodes[N - 2].first;

nodes[N - 2].first = temp;

}

for (i = 0; i < N; i++) //создали массив узлов в квадратах растной сетки

{

point\_in\_net(nodes[i].first, nodes[i].second, &x, &y);

nodes\_net.push\_back(make\_pair(x, y));

}

int k;

k = (nodes\_net[0].second - nodes\_net[N - 1].second) / netsize; //кол-во квадратов растной сетки по y (на каждом шаге спускаемся вниз на 1 по y ~ строка)

k++;

vector <active\_edge> active\_edges;

ycur = nodes\_net[0].second; //начальный y

ymin = nodes\_net[N - 1].second; //конечный y

w.push\_back(make\_pair(nodes\_net[0].first, nodes\_net[0].second)); //первый узел

do

{

for (j = 0; j < nodes.size(); j++)

{

if (nodes\_net[j].second == ycur) //если на строке есть узел

{

switch (j) //в зависимости от номера узла

{

case 0:

if (abs(nodes[1].second - nodes[0].second) < netsize / 2)

{

w.push\_back(make\_pair(nodes\_net[1].first, nodes\_net[1].second)); //если ребро горизонтально, добавили вторую вершину в массив частей растовой сетки

}

else

{

//иначе первая строка описывается только одной вершиной

w.push\_back(make\_pair(nodes\_net[0].first, nodes\_net[0].second));

//создаем активное ребро от 0 до 1 вершины

//угловой коэффициент

a.k = (double)(nodes[1].second - nodes[0].second) / (double)(nodes[1].first - nodes[0].first);

//откуда идем

a.x = nodes[0].first;

a.y = nodes[0].second;

//куда

a.yk = nodes[1].second;

a.xk = nodes[1].first;

//приращение по y - идем вниз, каждый раз добавляя 1 часть растовой сетки

a.dy = -netsize;

//приращение по x = приращение по y / угловой коэффициент

a.dx = (int)((double)a.dy / a.k);

a.dy = -netsize;

//если приращение посчиталось больше, чем сама сторона

//например в случае почти горизонтального ребра

//уменьшить приращение до размера стороны

if (abs(a.dx) > abs(a.xk - a.x)) a.dx = a.xk - a.x;

//на какой строке ребро закончится

a.yfin = nodes\_net[1].second;

active\_edges.push\_back(a);

}

//создаем активное ребро от 0 до 2 вершины, в любом случае

a.k = (double)(nodes[2].second - nodes[0].second) / (double)(nodes[2].first - nodes[0].first);

a.x = nodes[0].first;

a.yk = nodes[2].second;

a.y = nodes[0].second;

a.xk = nodes[2].first;

a.dy = -netsize;

//приращение по x = приращение по y / угловой коэффициент

a.dx = (int)((double)a.dy / a.k);

a.dy = -netsize;

if (abs(a.dx) > abs(a.xk - a.x)) a.dx = a.xk - a.x;

a.yfin = nodes\_net[2].second;

active\_edges.push\_back(a);

break;

case 1:

case 2:

case 3:

//создаем активное ребро от 1 до 3 вершины или от 2 до 4 вершины или от 6 до 7 вершины

a.k = (double)(nodes[j + 2].second - nodes[j].second) / (double)(nodes[j + 2].first - nodes[j].first);

a.x = nodes[j].first;

a.yk = nodes[j + 2].second;

a.y = nodes[j].second;

a.xk = nodes[j + 2].first;

a.dy = -netsize;

//приращение по x = приращение по y / угловой коэффициент

a.dx = (int)((double)a.dy / a.k);

a.dy = -netsize;

if (abs(a.dx) > abs(a.xk - a.x)) a.dx = a.xk - a.x;

a.yfin = nodes\_net[j + 2].second;

active\_edges.push\_back(a);

break;

case 4:

//если нижнее ребро горизонтально

if (nodes[N - 2].second == nodes[N - 1].second)

{

w.push\_back(make\_pair(nodes[N - 2].first, nodes[N - 2].second));

}

else

{ //создаем активное ребро от 6 до 7 вершины

a.k = (double)(nodes[N - 1].second - nodes[N - 2].second) / (double)(nodes[N - 1].first - nodes[N - 2].first);

a.x = nodes[N - 2].first;

a.yk = nodes[N - 1].second;

a.y = nodes[N - 2].second;

a.xk = nodes[N - 1].first;

a.dy = -netsize;

//приращение по x = приращение по y / угловой коэффициент

a.dx = (int)((double)a.dy / a.k);

if (abs(a.dx) > abs(a.xk - a.x)) a.dx = a.xk - a.x;

a.yfin = nodes\_net[N - 1].second;

active\_edges.push\_back(a);

}

break;

}

}

}

//на строке

for (i = 0; i < active\_edges.size(); i++)

{

//посчитали новые координаты

active\_edges[i].x += active\_edges[i].dx;

active\_edges[i].y += active\_edges[i].dy;

//нашли часть растовой сетки, содержащий точку, добавили его

if (point\_in\_net(active\_edges[i].x, active\_edges[i].y, &x, &y)) w.push\_back(make\_pair(x, y));

}

//проходим по активным ребрам

k = active\_edges.size();

for (i = 0; i < k; i++)

{

//если какое-то ребро должно закончиться

if (y <= active\_edges[i].yfin)

{

//убрали его

if (active\_edges.size() == 1) active\_edges.erase(active\_edges.begin(), active\_edges.end());

else active\_edges.erase(active\_edges.begin() + i);

k = active\_edges.size();

i--;

}

}

ycur--;

} while (ycur > ymin - 1); //пока не дошли до нижней строки

//добавили последний узел еще раз

w.push\_back(make\_pair(nodes\_net[N - 1].first, nodes\_net[N - 1].second));

//отсортировали по x внутри строк

sort(w.begin(), w.end(), group);

//K - кол-во добавляемых квадратов растной сетки

K = w.size();

//добавили квадраты растной сетки текущей фигуры в общий массив

for (i = 0; i < K; i++)

{

(\*v).push\_back(w[i]);

}

w.erase(w.begin(), w.end());

return (K);

}

void Process\_Mouse\_Move(int x, int y)

{

point p = point((GLint)x, height - (GLint)y);

if (cen\_ver.cent\_stat == true) //если центр установлен

{

cen\_ver.vertex\_x = p.x;

cen\_ver.vertex\_y = p.y;

cen\_ver.cord\_stat = true; //разрешить расчет координат шестиугольника

}

glutPostRedisplay();

}

void Process\_Mouse(int button, int state, int x, int y)

{

if (state != GLUT\_DOWN) return; //если кнопка мыши не была отпущена

if (button == GLUT\_LEFT\_BUTTON)

{

if (dot\_mode)

Change\_Mode();//отключить режим редактирования точек

if (!drawing\_mode)

drawing\_mode = true;

point p = point((GLint)x, height - (GLint)y);

if (cen\_ver.cent\_stat == false) //если центр еще не был установлен

{

current\_figure = -1;

cen\_ver.center\_x = p.x;

cen\_ver.center\_y = p.y;

cen\_ver.cent\_stat = true; //установить центр

}

else

{

cen\_ver.vertex\_x = p.x;

cen\_ver.vertex\_y = p.y;

cen\_ver.cent\_stat = false; //координаты определены

cen\_ver.cord\_stat = false; //и рассчитаны

Finish\_Primitive();

}

}

}

void Display(void)

{

int i, j, k;

glClear(GL\_COLOR\_BUFFER\_BIT);

glEnable(GL\_POLYGON\_SMOOTH);

glEnable(GL\_LINE\_SMOOTH);

glEnable(GL\_POINT\_SMOOTH);

if (colour\_shuffle\_mode) //если включен режим смешения цветов

{

glEnable(GL\_COLOR\_LOGIC\_OP);

if (colour\_shuffle\_mode == 1) glLogicOp(GL\_AND);

if (colour\_shuffle\_mode == 2) glLogicOp(GL\_NAND);

}

vector <pair<int, int>> net\_line; //массив сеткелей

if (raster\_mode == 1 || raster\_mode == 2) //построить контуры фигур

{

raster\_size.push\_back(0);

k = 0;

for (i = 0; i < v.size(); i++)

{

for (int j = 0; j < v[i].coord\_polygon.size(); j++)

{

//каждое ребро - отрезок

if (j != N - 1) k += contour\_Bresenham(v[i].coord\_polygon[j].x + v[i].Ox, v[i].coord\_polygon[j].y + v[i].Oy, v[i].coord\_polygon[j + 1].x + v[i].Ox, v[i].coord\_polygon[j + 1].y + v[i].Oy, &net\_line, v[i].Ox, v[i].Oy);

//+последняя сторона от 0 до 6 координаты

if (j == N - 1) k += contour\_Bresenham(v[i].coord\_polygon[N - 1].x + v[i].Ox, v[i].coord\_polygon[N - 1].y + v[i].Oy, v[i].coord\_polygon[0].x + v[i].Ox, v[i].coord\_polygon[0].y + v[i].Oy, &net\_line, v[i].Ox, v[i].Oy);

}

raster\_size.push\_back(k);

}

}

if (raster\_mode == 3) //построить заливку фигур

{

k = 0;

raster\_size.push\_back(0);

for (i = 0; i < v.size(); i++)

{

k += rasterize(v[i], &net\_line);

raster\_size.push\_back(k);

}

}

for (i = 0; i < v.size(); i++)

{

glPushMatrix(); //сохранить текущую матрицу

glTranslatef(v[i].Ox, v[i].Oy, 0); //сдвиг от начала координат

glScalef(v[i].ScaleOx, v[i].ScaleOy, 1); //масштабирование вдоль осей

glRotatef(v[i].Angle, 0, 0, 1); //вращение фигуры

if (v[i].texture\_number != -1) //если шестиугольник с текстурой

{

switch (v[i].overlay\_type\_tex) //определить тип наложения текстуры

{

case 1:

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);//результирующий цвет получается умножением

break;

case 2:

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL);//цвет объекта не учитывается

break;

}

glBindTexture(GL\_TEXTURE\_2D, names\_tex[v[i].texture\_number]); //выбрать текстуру

glEnable(GL\_TEXTURE\_2D); //разрешить текстурирование

}

//отрисовать шестиугольник

glBegin(GL\_POLYGON);

for (j = 0; j < v[i].coord\_polygon.size(); j++)

{

glColor3ub(v[i].color\_point[j].red, v[i].color\_point[j].green, v[i].color\_point[j].blue);

glTexCoord2f(texture\_coord[j].x, texture\_coord[j].y);

glVertex2i(v[i].coord\_polygon[j].x, v[i].coord\_polygon[j].y);

}

glEnd();

if (v[i].texture\_number != -1)

{

glDisable(GL\_TEXTURE\_2D);//запретить текстурирование

}

//обвести зеленым цветом (текущий)

if (v[i].number == current\_figure)

{

if (!raster\_mode)

{

glBegin(GL\_LINE\_LOOP);

glLineWidth(3.0);

glColor3ub(0, 240, 40);

for (j = 0; j < v[i].coord\_polygon.size(); j++)

{

glVertex2i(v[i].coord\_polygon[j].x, v[i].coord\_polygon[j].y);

}

glEnd();

}

if (dot\_mode && !raster\_mode)//если включен режим редактирования точек, выделить текущую точку красным

{

glPointSize(10);

glColor3ub(225, 0, 0);

glBegin(GL\_POINTS);

glVertex2i(v[i].coord\_polygon[current\_dot].x, v[i].coord\_polygon[current\_dot].y);

glEnd();

}

}

glPopMatrix(); //восстановить матрицу

//растеризируем многоугольники на сетке в режиме "контур"

if (raster\_mode == 1)

{

glBegin(GL\_QUADS);

for (k = raster\_size[i]; k < raster\_size[i + 1]; k++)

{

glColor3ub(30, 30, 30);

glVertex2i(netsize \* (net\_line[k].first - 1), netsize \* (net\_line[k].second - 1));

glVertex2i(netsize \* (net\_line[k].first), netsize \* (net\_line[k].second - 1));

glVertex2i(netsize \* (net\_line[k].first), netsize \* (net\_line[k].second));

glVertex2i(netsize \* (net\_line[k].first - 1), netsize \* (net\_line[k].second));

}

glEnd();

glPushMatrix(); //сохранить текущую матрицу

glTranslatef(v[i].Ox, v[i].Oy, 0); //сдвиг от начала координат

glScalef(v[i].ScaleOx, v[i].ScaleOy, 1); //масштабирование вдоль осей

glRotatef(v[i].Angle, 0, 0, 1); //вращение фигуры

glBegin(GL\_LINE\_LOOP);

glLineWidth(3.0);

glColor3ub(252, 48, 58);

for (j = 0; j < v[i].coord\_polygon.size(); j++)

{

glVertex2i(v[i].coord\_polygon[j].x, v[i].coord\_polygon[j].y);

}

glEnd();

glPopMatrix(); //восстановить матрицу

}

if (raster\_mode == 2)

{

for (j = 0; j < raster\_size.size() - 1; j++)

{

glColor3ub(v[j].color\_point[0].red, v[j].color\_point[0].green, v[j].color\_point[0].blue);

sort(net\_line.begin() + raster\_size[j], net\_line.begin() + raster\_size[j + 1], pred);

sort(net\_line.begin() + raster\_size[j], net\_line.begin() + raster\_size[j + 1], group);

for (k = raster\_size[j]; k < raster\_size[j + 1] - 1; k++)

{

glBegin(GL\_QUADS);

int k2;

//идем по "строкам" y, заполняя только нужную часть на строке

for (k2 = net\_line[k].first; k2 <= net\_line[k + 1].first; k2++)

{

glVertex2i(netsize \* (k2 - 1), netsize \* (net\_line[k].second - 1));

glVertex2i(netsize \* (k2), netsize \* (net\_line[k].second - 1));

glVertex2i(netsize \* (k2), netsize \* (net\_line[k].second));

glVertex2i(netsize \* (k2 - 1), netsize \* (net\_line[k].second));

}

glEnd();

}

}

}

//растеризируем многоугольники на сетке в режиме "заливка"

if (raster\_mode == 3)

{

for (k = raster\_size[i]; k < raster\_size[i + 1] - 1; k++)

{

glColor3ub(v[i].color\_point[0].red, v[i].color\_point[0].green, v[i].color\_point[0].blue);

if (net\_line[k].second == net\_line[k + 1].second)

{

glBegin(GL\_QUADS);

//идем по "строкам" y, заполняя только нужную часть на строке

for (j = net\_line[k].first; j <= net\_line[k + 1].first; j++)

{

glVertex2i(netsize \* (j - 1), netsize \* (net\_line[k].second - 1));

glVertex2i(netsize \* (j), netsize \* (net\_line[k].second - 1));

glVertex2i(netsize \* (j), netsize \* (net\_line[k].second));

glVertex2i(netsize \* (j - 1), netsize \* (net\_line[k].second));

}

glEnd();

}

}

}

//конец растеризации

}

if (drawing\_mode)

{

if (cen\_ver.cord\_stat == true) //если можно рассчитывать координаты

{

point cen = point(cen\_ver.center\_x, cen\_ver.center\_y);

point ver = point(cen\_ver.vertex\_x, cen\_ver.vertex\_y);

if (!equality(cen, ver)) //если координаты центра и вершины не совпадают

{

glPushMatrix(); //сохранить текущую матрицу

glTranslatef(cen.x, cen.y, 0); //переместить координаты в установленный центр

calc\_polygon\_coord(cen, ver); //рассчитать координаты шестиугольника

//отрисовать шестиугольник

glBegin(GL\_POLYGON);

glColor3ub(drawing\_color.red, drawing\_color.green, drawing\_color.blue);

for (i = 0; i < drawing\_v.size(); i++)

glVertex2i(drawing\_v[i].x, drawing\_v[i].y);

glEnd();

//обвести красным цветом (отрисовываемый)

glLineWidth(3.0);

glColor3ub(225, 0, 0);

glBegin(GL\_LINE\_LOOP);

for (i = 0; i < drawing\_v.size(); i++)

{

glVertex2i(drawing\_v[i].x, drawing\_v[i].y);

}

glEnd();

glPopMatrix(); //восстановить матрицу

}

}

}

//растеризация на сетке

if (raster\_mode) //включить сетку

{

glLineWidth(0.5);

glColor3ub(70, 70, 70);

glBegin(GL\_LINES);

for (i = 0; i < width / netsize + 1; i++)

{

glVertex2i(i\*netsize, 0);

glVertex2i(i\*netsize, height);

}

for (j = 0; j < height / netsize + 1; j++)

{

glVertex2i(0, j\*netsize);

glVertex2i(width, j\*netsize);

}

glEnd();

}

//очистить массивы сеткелей

net\_line.erase(net\_line.begin(), net\_line.end());

raster\_size.erase(raster\_size.begin(), raster\_size.end());

//если включен режим смешения цветов, отключить его

if (colour\_shuffle\_mode != 0) glDisable(GL\_COLOR\_LOGIC\_OP);

glFinish();

}

void Process\_Normal\_Keys(unsigned char key, int x, int y)

{

if (drawing\_mode);

if (!drawing\_mode && max\_figure != 0)

{

if (key == 32) //пробел

{

Change\_Mode(); //включить/отключить режим редактирования точек

}

if (key == 127) //delete

{

Del\_Group();

}

if (!dot\_mode)

{

if (key == 'w' || key == 'W')

{

MoveUp();

}

if (key == 's' || key == 'S')

{

MoveDown();

}

if (key == 'a' || key == 'A')

{

MoveLeft();

}

if (key == 'd' || key == 'D')

{

MoveRight();

}

if (key == '+')

{

Increase(0.05);

}

if (key == '-')

{

Reduce(0.05);

}

if (key == 'l' || key == 'L')

{

SwitchColourShuffleMode();

}

}

}

if (!drawing\_mode && !dot\_mode)

{

if (key == 'n' || key == 'N')

{

SwitchRasterMode();

}

}

if (!drawing\_mode && !dot\_mode && raster\_mode)

{

if (key == 'm' || key == 'M')

{

SwitchNetSize();

}

}

if (key == 'p' || key == 'P')

{

Clear();

}

glutPostRedisplay();

}

void Process\_Special\_Keys(int key, int x, int y)

{

if (!drawing\_mode && max\_figure != 0)

{

if (key == GLUT\_KEY\_UP)

{

On\_Up\_Key();

}

if (key == GLUT\_KEY\_DOWN)

{

On\_Down\_Key();

}

if (!dot\_mode)

{

if (key == GLUT\_KEY\_LEFT)

{

RotateLeft();

}

if (key == GLUT\_KEY\_RIGHT)

{

RotateRight();

}

}

}

glutPostRedisplay();

}

void Initialize()

{

//установить белый цвет экрана

glClearColor(1.0, 1.0, 1.0, 1);

//установить проекцию

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glViewport(0, 0, width, height);

gluOrtho2D(0.0, width, 0.0, height);

//установить начальные значения

drawing\_mode = false;

dot\_mode = false;

max\_figure = 0;

current\_figure = 0;

current\_dot = 0;

cen\_ver.cent\_stat = false; //центр отрисовываемого шестиугольника не установлен

cen\_ver.cord\_stat = false; //расчет координат шестиугольника запрещен

raster\_mode = 0; //по умолчанию сетка отключена

netsize = 6; //размер сетки по умолчанию

glutSetMenu(main\_menu1); //установить основное меню

loadTexture(); //загрузить текстуры

set\_texture\_coord();//рассчитать текстурные координаты

colour\_shuffle\_mode = 0; //по умолчанию нет смешения цветов

}

void Reshape(GLint w, GLint h)

{

//изменить размеры окна

width = w;

height = h;

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glViewport(0, 0, width, height);

gluOrtho2D(0.0, width, 0.0, height);

}

Lab2.cpp

#include "pch.h"

#include <iostream>

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); //один буфер отрисовки окна, использование всех компонент RGB

glutInitWindowSize(1200, 800);

glutCreateWindow("Lab2 Octogon");

Initialize(); //начальные установки

createMenu(); //создание меню

glutKeyboardFunc(Process\_Normal\_Keys); //обработка клавиш с кодами ascii

glutSpecialFunc(Process\_Special\_Keys); //обработка не-ascii клавиш

glutReshapeFunc(Reshape);

glutDisplayFunc(Display);

glutMouseFunc(Process\_Mouse);

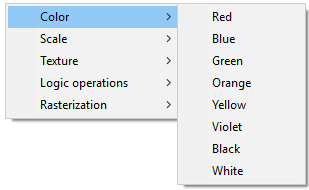
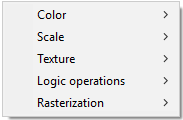
glutPassiveMotionFunc(Process\_Mouse\_Move);

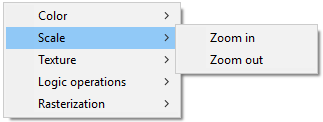
glutMainLoop();

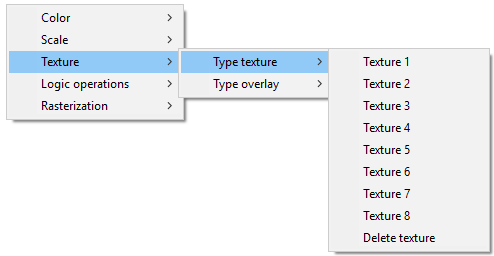
}

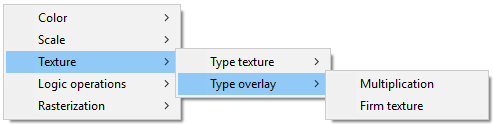
1. Тесты

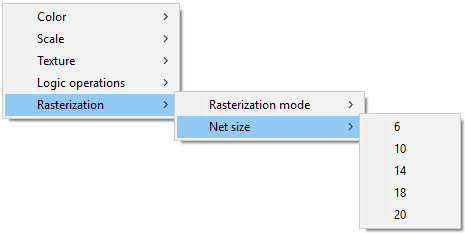
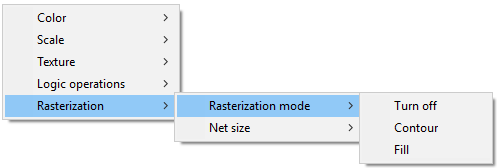
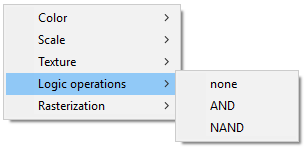
Тест 1. Пример меню.



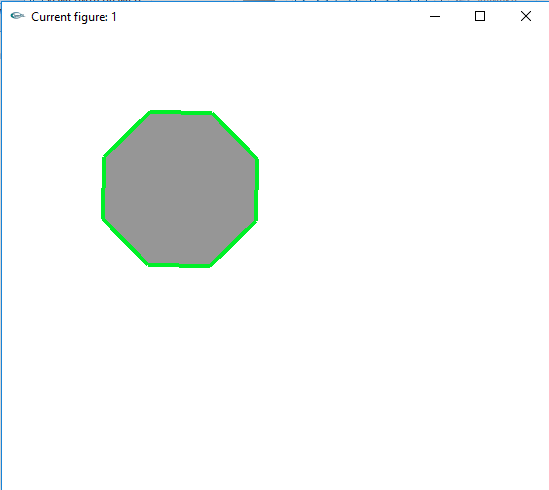




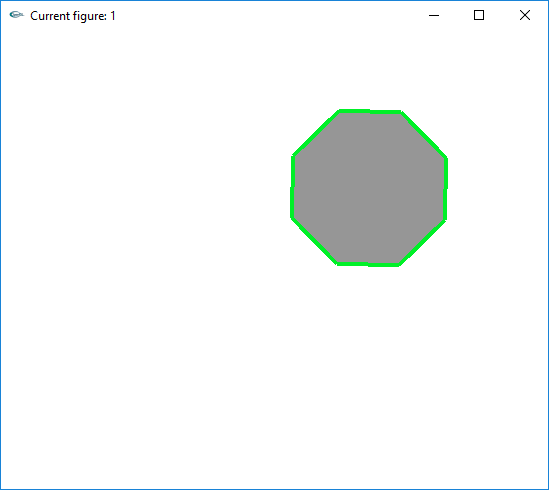
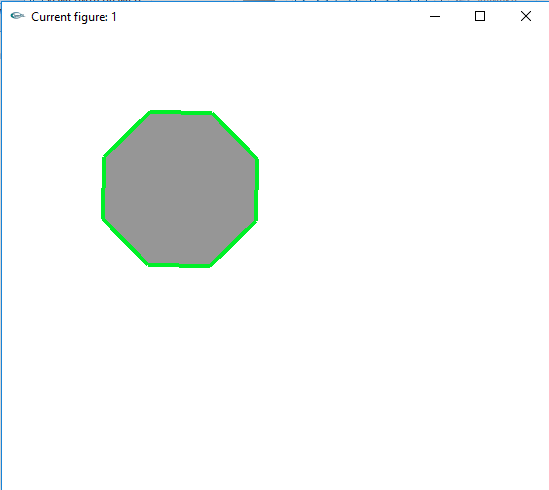




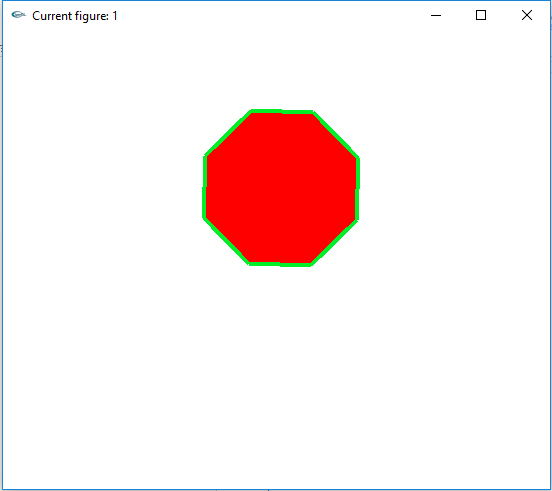
Тест 2. Пример восьмиугольника.



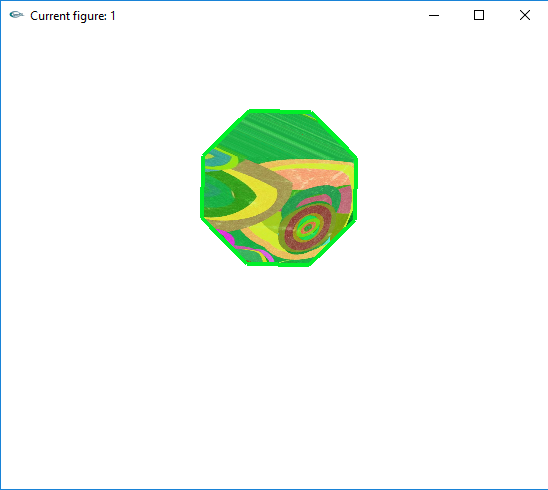
Тест 3. Движение восьмиугольника



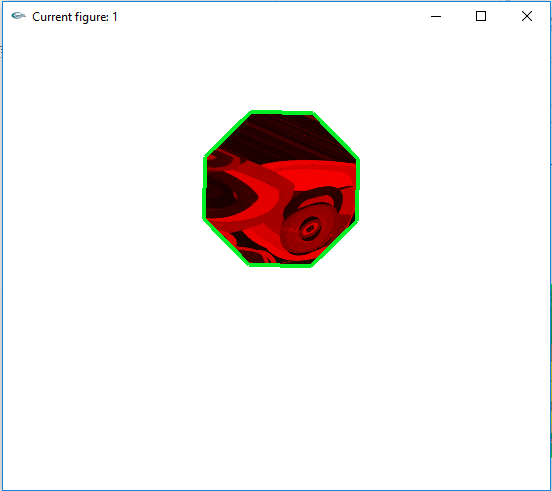
Тест 4. Изменение цвета.



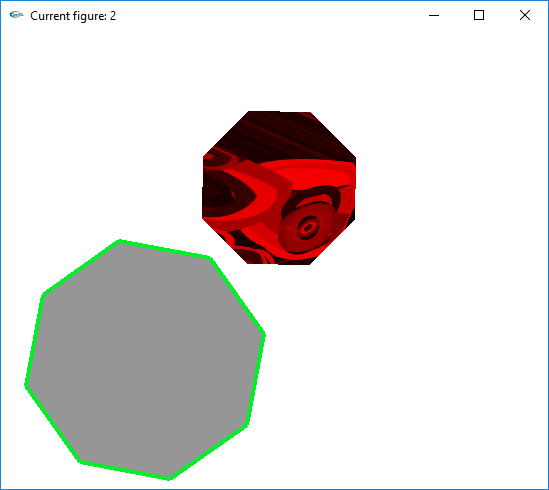
Тест 5. Изменение текстуры.



С учётом цвета фигуры.



Тест 6. Добавление нескольких фигур.



Тест 7. Удаление восьмиугольника.

