Оглавление

[Задание 2](#_Toc5113527)

[Исходный код 3](#_Toc5113528)

[main.cpp 3](#_Toc5113529)

[julia.h 3](#_Toc5113530)

[julia.cpp 4](#_Toc5113531)

[myBitmap.h 5](#_Toc5113532)

[myBitmap.cpp 6](#_Toc5113533)

[colors.h 8](#_Toc5113534)

[colors.cpp 8](#_Toc5113535)

[Тестирование 9](#_Toc5113536)

[Проверка 10](#_Toc5113537)

# Задание

Компьютерная графика

Лабораторная работа № 1 «Фрактальная графика»

Задание:

Реализовать программу, строящую фрактал с заданными границами расчета

.

вид фрактала и границы расчета

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вид фрактала |  |  |  |  |
| Джулия | -0.9 | 0.9 | -1.1 | 1.0 |

Программа на языке Visual C++ (без применения openGL)

# Исходный код

## main.cpp

#include <iostream>

#include <cstdlib>

#include "julia.h"

#include "myBitmap.h"

using namespace std;

int main()

{

julia j;

/\* формирование фрактала \*/

j.draw(0, 0, 1000, 1000, -0.9, 0.9, -1.1, 1.0);

system("PAUSE");

return EXIT\_SUCCESS;

}

## julia.h

#pragma once

#include "myBitmap.h"

class julia

{

public:

julia();

~julia();

/\* процедура формирования фрактала \*/

void draw(int xx, int yy, int cx, int cy, double minx, double maxx, double miny, double maxy);

/\* определение цвета точки \*/

int c(int index);

/\* функция подсчета количества итераций\*/

int iteration(double x, double y);

private:

myBitmap bmp;

};

## julia.cpp

#include "julia.h"

#include <cmath>

#include <iostream>

#include "colors.h"

#define mi 511

using namespace std;

julia::julia()

{

}

julia::~julia()

{

}

/\* процедура формирования фрактала \*/

void julia::draw(int xx, int yy, int cx, int cy, double minx, double maxx, double miny, double maxy)

{

bmp.create(cx, cy);

DWORD\* bits = bmp.bits();

int pos;

double stepx, stepy, x, y;

int i, j, iter;

stepx = (maxx - minx) / cx;

stepy = (maxy - miny) / cy;

y = miny;

for (j = 0; j < cy; ++j) // for j := 0 to cy

{

pos = j \* cx;

x = minx;

for (i = 0; i < cx; ++i) // for i := 0 to cx

{

iter = iteration(x, y);

bits[pos++] = COLORS[iter % COLORS\_COUNT]; // putpixel(xx + i, yy + j, c(iter));

x += stepx;

}

y += stepy;

} // j

bmp.saveBitmap("julia.bmp");

}

/\* определение цвета точки \*/

int julia::c(int index)

{

return 7 \* (mi - index);

}

/\* функция подсчета количества итераций \*/

int julia::iteration(double x, double y)

{

const double cx = 0.36;

const double cy = 0.36;

int i = 0;

double xx = x, yy = y, xk, yk;

while (pow(xx, 2.0) + pow(yy, 2.0) <= 4.0)

{

xk = pow(xx, 2.0) - pow(yy, 2.0) + cx;

yk = 2 \* xx \* yy + cy;

xx = xk;

yy = yk;

++i;

if (i >= mi) break;

}

return i;

}

## myBitmap.h

#pragma once

#include <string>

#include <windows.h>

class myBitmap

{

public:

myBitmap();

~myBitmap();

bool create(int w, int h);

void clear(BYTE clr = 0);

void setBrushColor(DWORD bClr);

void setPenColor(DWORD c);

void setPenWidth(int w);

void saveBitmap(std::string path);

HDC getDC() const;

int getWidth() const;

int getHeight() const;

DWORD\* bits() const;

private:

void createPen();

HBITMAP bmp;

HDC hdc;

HPEN pen;

HBRUSH brush;

void\* pBits;

int width, height, wid;

DWORD clr;

};

## myBitmap.cpp

#include "myBitmap.h"

myBitmap::myBitmap() : pen(NULL), brush(NULL), clr(0), wid(1)

{

}

myBitmap::~myBitmap()

{

DeleteObject(pen);

DeleteObject(brush);

DeleteDC(hdc);

DeleteObject(bmp);

}

bool myBitmap::create(int w, int h)

{

BITMAPINFO bi;

ZeroMemory(&bi, sizeof(bi));

bi.bmiHeader.biSize = sizeof(bi.bmiHeader);

bi.bmiHeader.biBitCount = sizeof(DWORD)\* 8;

bi.bmiHeader.biCompression = BI\_RGB;

bi.bmiHeader.biPlanes = 1;

bi.bmiHeader.biWidth = w;

bi.bmiHeader.biHeight = -h;

HDC dc = GetDC(GetConsoleWindow());

bmp = CreateDIBSection(dc, &bi, DIB\_RGB\_COLORS, &pBits, NULL, 0);

if (!bmp) return false;

hdc = CreateCompatibleDC(dc);

SelectObject(hdc, bmp);

ReleaseDC(GetConsoleWindow(), dc);

width = w; height = h;

return true;

}

void myBitmap::clear(BYTE clr)

{

memset(pBits, clr, width \* height \* sizeof(DWORD));

}

void myBitmap::setBrushColor(DWORD bClr)

{

if (brush) DeleteObject(brush);

brush = CreateSolidBrush(bClr);

SelectObject(hdc, brush);

}

void myBitmap::setPenColor(DWORD c)

{

clr = c; createPen();

}

void myBitmap::setPenWidth(int w)

{

wid = w; createPen();

}

void myBitmap::saveBitmap(std::string path)

{

BITMAPFILEHEADER fileheader;

BITMAPINFO infoheader;

BITMAP bitmap;

DWORD wb;

GetObject(bmp, sizeof(bitmap), &bitmap);

DWORD\* dwpBits = new DWORD[bitmap.bmWidth \* bitmap.bmHeight];

ZeroMemory(dwpBits, bitmap.bmWidth \* bitmap.bmHeight \* sizeof(DWORD));

ZeroMemory(&infoheader, sizeof(BITMAPINFO));

ZeroMemory(&fileheader, sizeof(BITMAPFILEHEADER));

infoheader.bmiHeader.biBitCount = sizeof(DWORD)\* 8;

infoheader.bmiHeader.biCompression = BI\_RGB;

infoheader.bmiHeader.biPlanes = 1;

infoheader.bmiHeader.biSize = sizeof(infoheader.bmiHeader);

infoheader.bmiHeader.biHeight = bitmap.bmHeight;

infoheader.bmiHeader.biWidth = bitmap.bmWidth;

infoheader.bmiHeader.biSizeImage = bitmap.bmWidth \* bitmap.bmHeight \* sizeof(DWORD);

fileheader.bfType = 0x4D42;

fileheader.bfOffBits = sizeof(infoheader.bmiHeader) + sizeof(BITMAPFILEHEADER);

fileheader.bfSize = fileheader.bfOffBits + infoheader.bmiHeader.biSizeImage;

GetDIBits(hdc, bmp, 0, height, (LPVOID)dwpBits, &infoheader, DIB\_RGB\_COLORS);

HANDLE file = CreateFile(path.c\_str(), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL, NULL);

WriteFile(file, &fileheader, sizeof(BITMAPFILEHEADER), &wb, NULL);

WriteFile(file, &infoheader.bmiHeader, sizeof(infoheader.bmiHeader), &wb, NULL);

WriteFile(file, dwpBits, bitmap.bmWidth \* bitmap.bmHeight \* 4, &wb, NULL);

CloseHandle(file);

delete[] dwpBits;

}

HDC myBitmap::getDC() const

{

return hdc;

}

int myBitmap::getWidth() const

{

return width;

}

int myBitmap::getHeight() const

{

return height;

}

DWORD\* myBitmap::bits() const

{

return (DWORD\*)pBits;

}

void myBitmap::createPen() {

if (pen) DeleteObject(pen);

pen = CreatePen(PS\_SOLID, wid, clr);

SelectObject(hdc, pen);

}

## colors.h

#pragma once

const int COLORS\_COUNT = 21;

extern int COLORS[COLORS\_COUNT];

## colors.cpp

#include "colors.h"

int COLORS[COLORS\_COUNT] = {

0xFFFFB300, //Vivid Yellow

0xFF803E75, //Strong Purple

0xFFFF6800, //Vivid Orange

0xFFA6BDD7, //Very Light Blue

0xFFC10020, //Vivid Red

0xFFCEA262, //Grayish Yellow

0xFF817066, //Medium Gray

0xFF007D34, //Vivid Green

0xFFF6768E, //Strong Purplish Pink

0xFF00538A, //Strong Blue

0xFFFF7A5C, //Strong Yellowish Pink

0xFF53377A, //Strong Violet

0xFFFF8E00, //Vivid Orange Yellow

0xFFB32851, //Strong Purplish Red

0xFFF4C800, //Vivid Greenish Yellow

0xFF7F180D, //Strong Reddish Brown

0xFF93AA00, //Vivid Yellowish Green

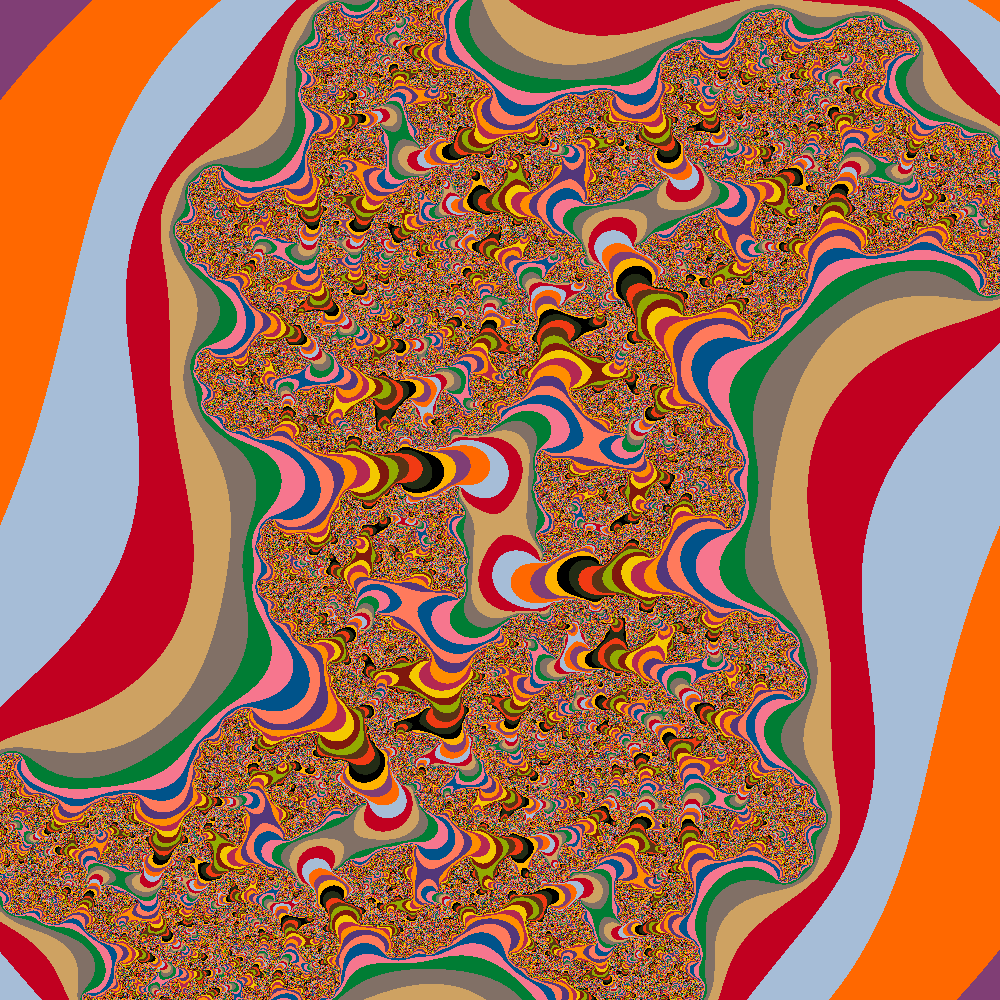
0xFF593315, //Deep Yellowish Brown

0xFFF13A13, //Vivid Reddish Orange

0xFF232C16 //Dark Olive Green

};

# Тестирование



# Проверка

