Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

Практическая работа №6

«Вывод растровых изображений в OpenGL»

ДИСЦИПЛИНА: «Компьютерная графика»

Выполнил: студент гр. ИУК5-42Б _____ (____Ли Р. В.____)
(Подпись) (Ф.И.О.)

Проверил: _____ (___Широкова Е. В.____)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024 г.

Цели: формирование практических навыков по работе с растровыми изображениями средствами OpenGL, а также простейшим преобразованиям цветовых и пространственных характеристик.

Задачи: понимать принципы вывода растровых изображений, знать отличия битового образа от растрового и их характеристики, научиться использовать средства OpenGL для вывода растровых изображений, знать основные константы OpenGL, используемые при обработке растровых изображений, уметь создавать приложения OpenGL с использованием функций для работы с растровыми изображениями.

Задание выполняется согласно варианту. По завершении готовится отчет.

1) Для Листинга 1 создать битовые образы согласно варианту
2) Для Листингов 2, 3, 4 осуществить вывод изображения в формате tga согласно варианту
3) Для Листинга 5 реализовать преобразование изображения согласно варианту.

Вариант №8

tga.h:

```
#include <vector>
#include <fstream>
#include <cstring>
#include <cstdint>

typedef union PixelInfo {
	std::uint32_t Colour;
	struct {
		std::uint8_t R, G, B, A;
	};
} *PPixelInfo;

class Tga {
public:
	Tga(const char* FilePath);
	std::vector<std::uint8_t> GetPixels() { return this->Pixels; }
	std::uint32_t GetWidth() const { return this->width; }
	std::uint32_t GetHeight() const { return this->height; }
	bool HasAlphaChannel() { return BitsPerPixel == 32; }
private:
	std::vector<std::uint8_t> Pixels;
	bool ImageCompressed;
	std::uint32_t width, height, size, BitsPerPixel;
};

Tga::Tga(const char* FilePath) {
	std::fstream hFile(FilePath, std::ios::in | std::ios::binary);
	if (!hFile.is_open()) { throw std::invalid_argument("File Not Found"); }
```

```
std::uint8_t Header[18] = { 0 };
std::vector<std::uint8_t> ImageData;
static std::uint8_t DeCompressed[12] = { 0x0, 0x0, 0x2, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0 };
static std::uint8_t IsCompressed[12] = { 0x0, 0x0, 0xA, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0 };
hFile.read(reinterpret_cast<char*>(&Header), sizeof(Header));
if (!std::memcmp(DeCompressed, &Header, sizeof(DeCompressed))) {
        BitsPerPixel = Header[16];
        width = Header[13] * 256 + Header[12];
        height = Header[15] * 256 + Header[14];
        size = ((width * BitsPerPixel + 31) / 32) * 4 * height;
        if ((BitsPerPixel != 24) && (BitsPerPixel != 32)) {
                hFile.close();
                throw std::invalid_argument("Invalid File Format. Required: 24 or 32 Bit Image");
        }
        ImageData.resize(size);
        ImageCompressed = false;
        hFile.read(reinterpret_cast<char*>(ImageData.data()), size);
}
else if (!std::memcmp(IsCompressed, &Header, sizeof(IsCompressed))) {
        BitsPerPixel = Header[16];
        width = Header[13] * 256 + Header[12];
        height = Header[15] * 256 + Header[14];
        size = ((width * BitsPerPixel + 31) / 32) * 4 * height;
        if ((BitsPerPixel != 24) && (BitsPerPixel != 32)) {
                hFile.close();
                throw std::invalid_argument("Invalid File Format. Required: 24 or 32 Bit Image");
        }
        PixelInfo Pixel = { 0 };
        int CurrentByte = 0;
        std::size_t CurrentPixel = 0;
        ImageCompressed = true;
        std::uint8_t ChunkHeader = { 0 };
        int BytesPerPixel = (BitsPerPixel / 8);
        ImageData.resize(width * height * sizeof(PixelInfo));
        do {
                hFile.read(reinterpret_cast<char*>(&ChunkHeader), sizeof(ChunkHeader));
                if (ChunkHeader < 128) {
                        for (int i{}; i < ChunkHeader; ++i, ++CurrentPixel) {
                                hFile.read(reinterpret_cast<char*>(&Pixel), BytesPerPixel);
                                ImageData[CurrentByte++] = Pixel.B;
                                ImageData[CurrentByte++] = Pixel.G;
                                ImageData[CurrentByte++] = Pixel.R;
                                if (BitsPerPixel > 24)
                                        ImageData[CurrentByte++] = Pixel.A;
                        }
                }
                else {
                        ChunkHeader -= 127;
                        hFile.read(reinterpret_cast<char*>(&Pixel), BytesPerPixel);
                        for (int i{}; i < ChunkHeader; ++i, ++CurrentPixel) {
                                ImageData[CurrentByte++] = Pixel.B;
                                ImageData[CurrentByte++] = Pixel.G;
                                ImageData[CurrentByte++] = Pixel.R;
                                if (BitsPerPixel > 24)
                                        ImageData[CurrentByte++] = Pixel.A;
                        }
                }
        } while (CurrentPixel < (width * height));
}
else {
        hFile.close();
```

```
                throw std::invalid_argument("Invalid File Format. Required: 24 or 32 Bit TGA File.");
        }
        hFile.close();
        this->Pixels = ImageData;
}
```

Задание №1

Формулировка задания:

Создать два разных битовых образа (не из примера) размерами 10x10 и 15x15. Отобразить ромб, в котором границей является образ №1, а заполнение происходит при помощи образа №2. Граница должна быть одного цвета, заполненные внутренние образы случайного цвета.

Листинг программы:

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

GLubyte first[32] = {0xff, 0xff,
   0x00, 0x00,
   0x00, 0x00,
   0x00, 0x00,

   0x3,  0xc0,
   0x7,  0xe0,
   0xf,  0xf0,
   0x1f, 0xf8,
   0x1f,  0xf8,
   0x1f, 0xf8,
   0x1f, 0xf8,
   0xe,  0x78,
   0xc, 0x30,

   0x00, 0x00,
   0x00, 0x00,
   0x00, 0x00};

GLubyte second[32] = {
   0x00, 0x00,

   0x0, 0x80,
   0x0, 0x80,
   0x1, 0xc0,
   0x1, 0xc0,
   0x1, 0xc0,
   0x3, 0xe0,
   0x1f, 0xfc,
   0x7f, 0xff,
   0x1f, 0xfc,
   0x3, 0xe0,
   0x1, 0xc0,
   0x1, 0xc0,
   0x1, 0xc0,
   0x0, 0x80,
   0x0, 0x80,
```

```
};

GLfloat randomValue() { return ((GLfloat)(rand() % 11)) * 0.1f; }

void filling(int x, int y, int size, float range) {
  int sizesAmount = (int)range / 16;
  for (int j = 0; j < sizesAmount; j++) {
    glColor3d(randomValue(), randomValue(), randomValue());
    if (j + 1 < (sizesAmount / 2) + 1) {
      glRasterPos2i(x - (0.5 + 1 + j) * size, y);
      glBitmap(size, size, 0.0, 0.0, 0.0, 0.0, first);
      continue;
    }
    if (j + 1 == (sizesAmount / 2) + 1) {
      glRasterPos2i(x - 0.5 * size, y);
      glBitmap(size, size, 0.0, 0.0, 0.0, 0.0, first);
      continue;
    }
    glRasterPos2i(x + (sizesAmount - j - 0.5) * size, y);
    glBitmap(size, size, 0.0, 0.0, 0.0, 0.0, first);
  }
}

void SetupRC() { glClearColor(0.0f, 0.0f, 0.0f, 0.0f); }

void ChangeSize(int w, int h) {
  if (h == 0)
    h = 1;
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(10, (GLfloat)w, 0.0f, (GLfloat)h);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void RenderScene(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
  glColor3f(1.0f, 1.0f, 1.0f);
  float x = 248, y = 136, size = 16, range = 16;
  for (int i = 0; i < 15; i++) {
    if (i == 0 || i == 14) {
      glRasterPos2i(x - size / 2, y);
      glBitmap(size, size, 0.0, 0.0, 0.0, 0.0, second);
      y += size;
      continue;
    }
    filling(x, y, size, range);
    glColor3f(1.0f, 1.0f, 1.0f);
    glRasterPos2i(x - range / 2 - size, y);
    glBitmap(size, size, 0.0, 0.0, 0.0, 0.0, second);
    glRasterPos2i(x + range / 2, y);
    glBitmap(size, size, 0.0, 0.0, 0.0, 0.0, second);
    y += size;
    if (i < 7) {
      range += 2 * size;
      continue;
    }
    range -= 2 * size;
  }
  glutSwapBuffers();
}
```

```
int main(int argc, char *argv[]) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
  glutInitWindowSize(512, 512);
  glutCreateWindow("Task 1");
  glutReshapeFunc(ChangeSize);
  glutDisplayFunc(RenderScene);
  SetupRC();
  glutMainLoop();
  return 0;
}
```

Результаты выполнения программы:

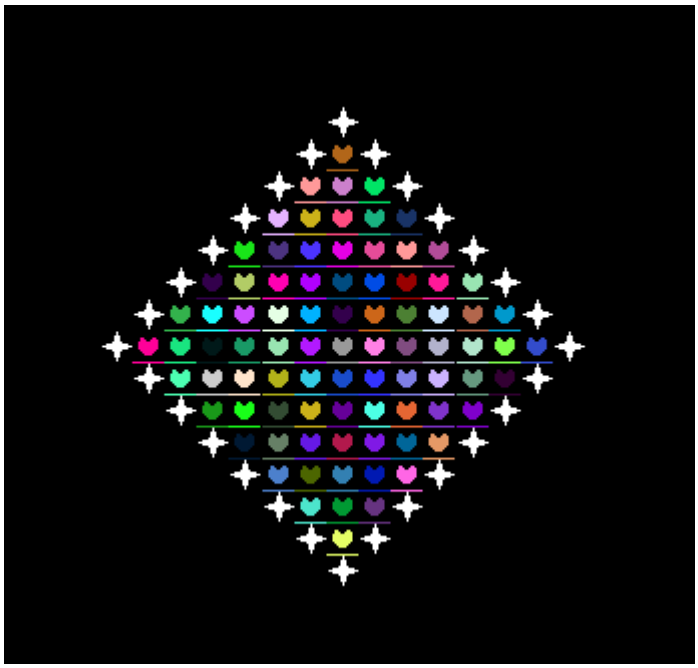

Рис. 1 Битовые образы

Задание №2

Формулировка задания:

Для собственного изображения формата tga, отобразить 10 его копий со случайными координатами.

Листинг программы:

```
#include "GL/freeglut.h"
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <algorithm>
```

```
#include "tga.h"

GLint Gw = 1600;
GLint Gh = 900;

std::vector<std::uint8_t> randomPixels(Tga image) {
	std::vector<std::uint8_t> newPixels(image.GetPixels());
	std::random_shuffle(newPixels.begin(), newPixels.end());
	return newPixels;
}

void display(void) {
	glClearColor(1, 1, 1, 1);
	glClear(GL_COLOR_BUFFER_BIT);
	Tga image = Tga("image.tga");
	for (int i = 0; i < 2; i++) {
		glRasterPos2i(160 + image.GetWidth() * i, 0);
		glDrawPixels(image.GetWidth(), image.GetHeight(), GL_RGB, GL_UNSIGNED_BYTE,
randomPixels(image).data());
	}
	glRasterPos2i(160, 474);
	glDrawPixels(image.GetWidth(), image.GetHeight(), GL_RGB, GL_UNSIGNED_BYTE,
image.GetPixels().data());
	glRasterPos2i(160 + image.GetWidth(), 474);
	glDrawPixels(image.GetWidth(), image.GetHeight(), GL_RGB, GL_UNSIGNED_BYTE,
randomPixels(image).data());
	glutSwapBuffers();
}

void ChangeSize(int w, int h) {
	if (h == 0) h = 1;
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
	gluOrtho2D(0, (GLfloat)w, 0, (GLfloat)h);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	Gw = w;
	Gh = h;
}

int main(int argc, char* argv[]) {
	srand(time(0));
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
	glutInitWindowSize(Gw, Gh);
	glutInitWindowPosition(0, 0);
	glutCreateWindow("Task 2");
	glutReshapeFunc(ChangeSize);
	glutDisplayFunc(display);
	glutMainLoop();
	return 0;
}
```

Результаты выполнения программы:

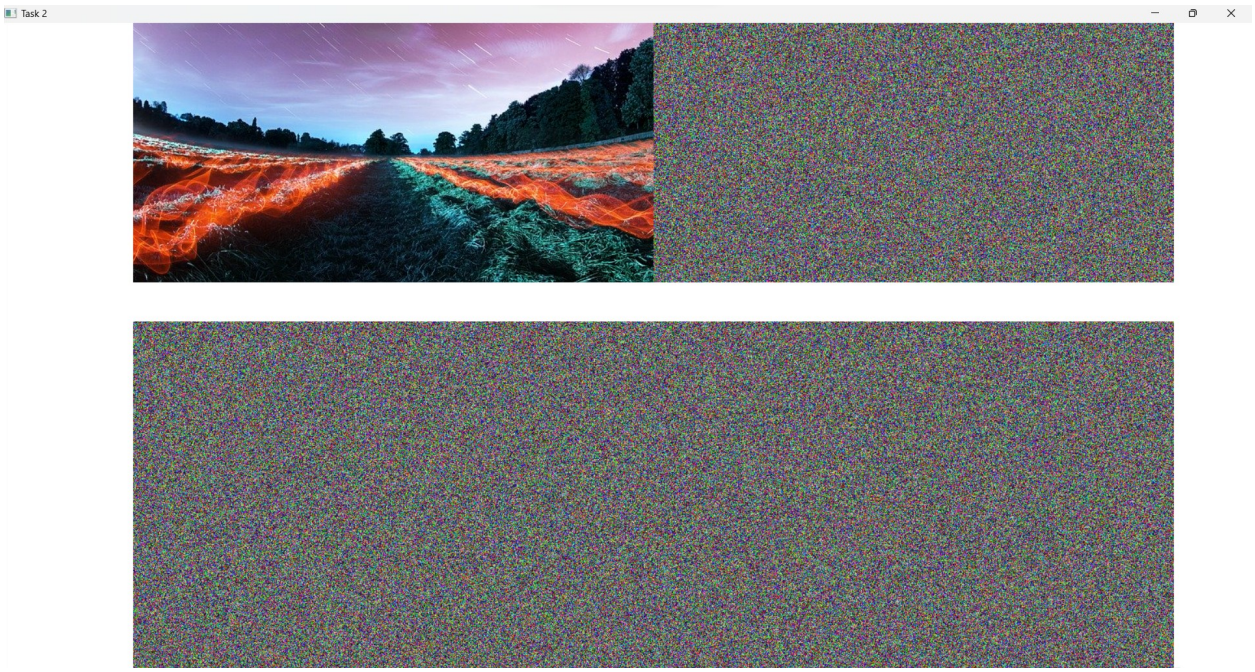

Рис. 2 Перемешивание пикселей

Задание №3

Формулировка задания:

Для собственного изображения формата tga выполнить случайное изменение компонент RGB при каждом запуске.

Листинг программы:

```
#include "GL/freeglut.h"
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <algorithm>
#include "tga.h"

GLint Gw = 1600;
GLint Gh = 900;

GLfloat randomValue() {
	return ((GLfloat)(rand() % 11)) * 0.1f;
}

void display(void) {
	glClearColor(1, 1, 1, 1);
	glClear(GL_COLOR_BUFFER_BIT);
	Tga image = Tga("image.tga");
```

```
	glRasterPos2i(480, 237);
	glDrawPixels(image.GetWidth(), image.GetHeight(), GL_RGB, GL_UNSIGNED_BYTE,
image.GetPixels().data());
	glutSwapBuffers();
}

void ChangeSize(int w, int h) {
	if (h == 0) h = 1;
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
	gluOrtho2D(0, (GLfloat)w, 0, (GLfloat)h);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	Gw = w;
	Gh = h;
}

void ProcessMenu(int value) {
	switch (value) {
	case 0:
		glPixelTransferf(GL_RED_SCALE, 1.0f);
		glPixelTransferf(GL_GREEN_SCALE, 0.0f);
		glPixelTransferf(GL_BLUE_SCALE, 0.0f);
		break;
	case 1:
		glPixelTransferf(GL_RED_SCALE, 0.0f);
		glPixelTransferf(GL_GREEN_SCALE, 1.0f);
		glPixelTransferf(GL_BLUE_SCALE, 0.0f);
		break;
	case 2:
		glPixelTransferf(GL_RED_SCALE, 0.0f);
		glPixelTransferf(GL_GREEN_SCALE, 0.0f);
		glPixelTransferf(GL_BLUE_SCALE, 1.0f);
		break;
	default:
		glPixelTransferf(GL_RED_SCALE, randomValue());
		glPixelTransferf(GL_GREEN_SCALE, randomValue());
		glPixelTransferf(GL_BLUE_SCALE, randomValue());
		break;
	}
	glutPostRedisplay();
}

int main(int argc, char* argv[]) {
	srand(time(0));
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
	glutInitWindowSize(Gw, Gh);
	glutInitWindowPosition(0, 0);
	glutCreateWindow("Task 3");
	glutReshapeFunc(ChangeSize);
	glutDisplayFunc(display);
	glutCreateMenu(ProcessMenu);
	glutAddMenuEntry("Just red", 0);
	glutAddMenuEntry("Just green", 1);
	glutAddMenuEntry("Just blue", 2);
	glutAddMenuEntry("Random", 3);
	glutAttachMenu(GLUT_RIGHT_BUTTON);
	glutMainLoop();
	return 0;
}
```

Результаты выполнения программы:

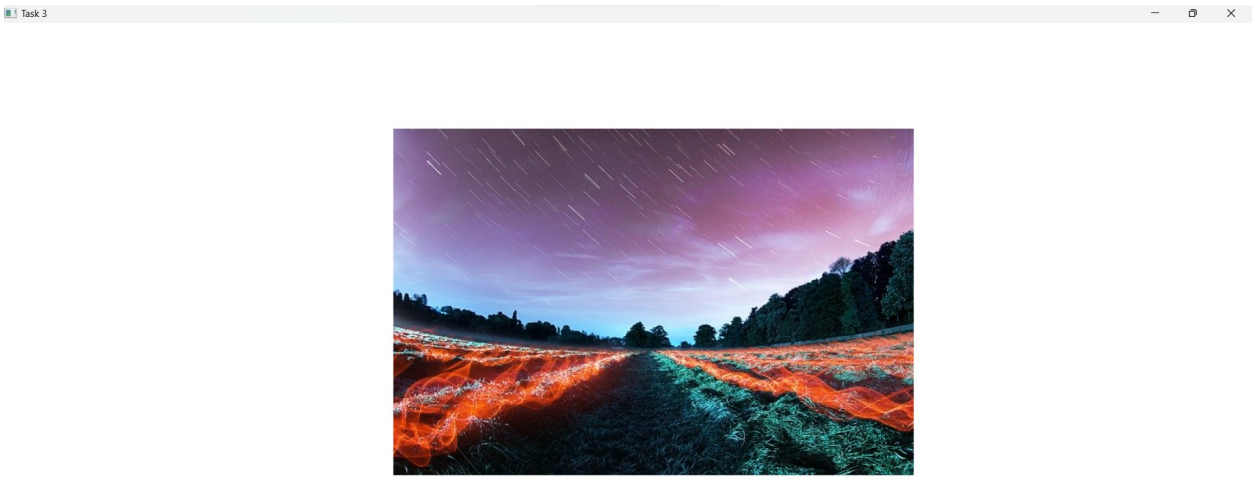

Рис. 3 Изображение без изменения каналов цвета

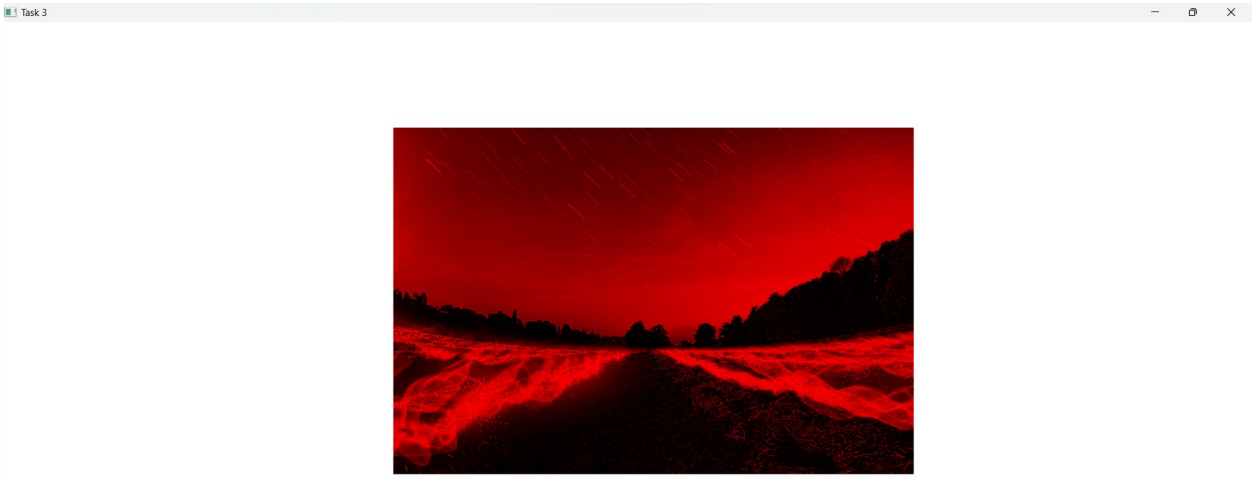

Рис. 4 Изображение, использующее только красный канал

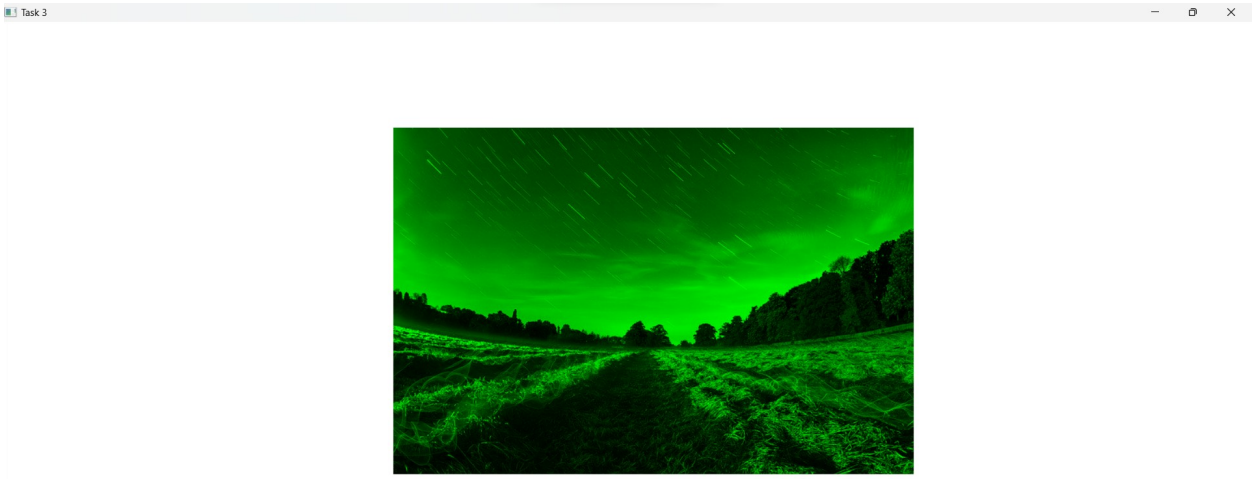

Рис. 5 Изображение, использующее только зелёный канал

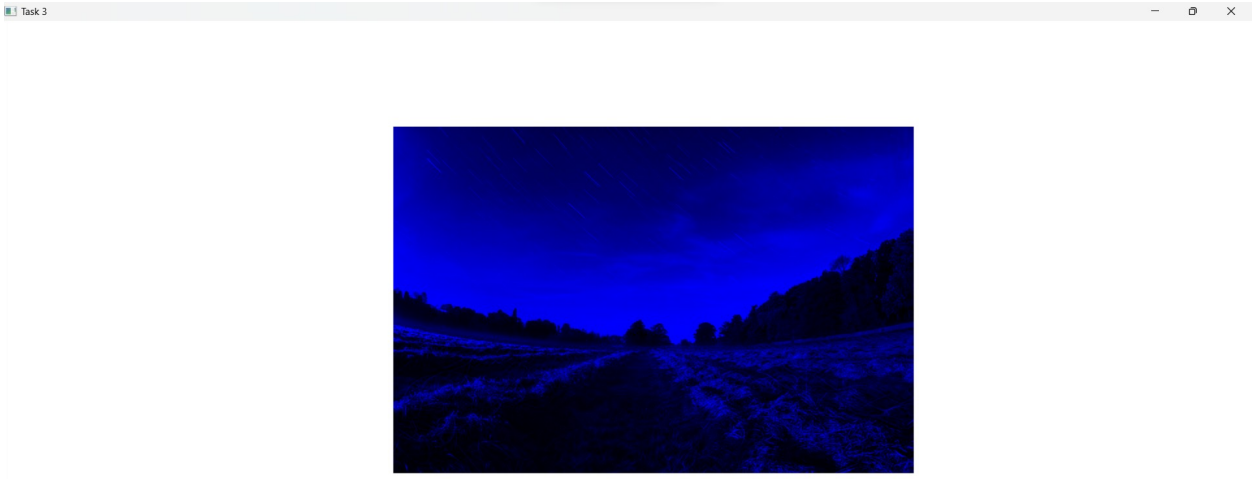

Рис. 6 Изображение, использующее только синий канал

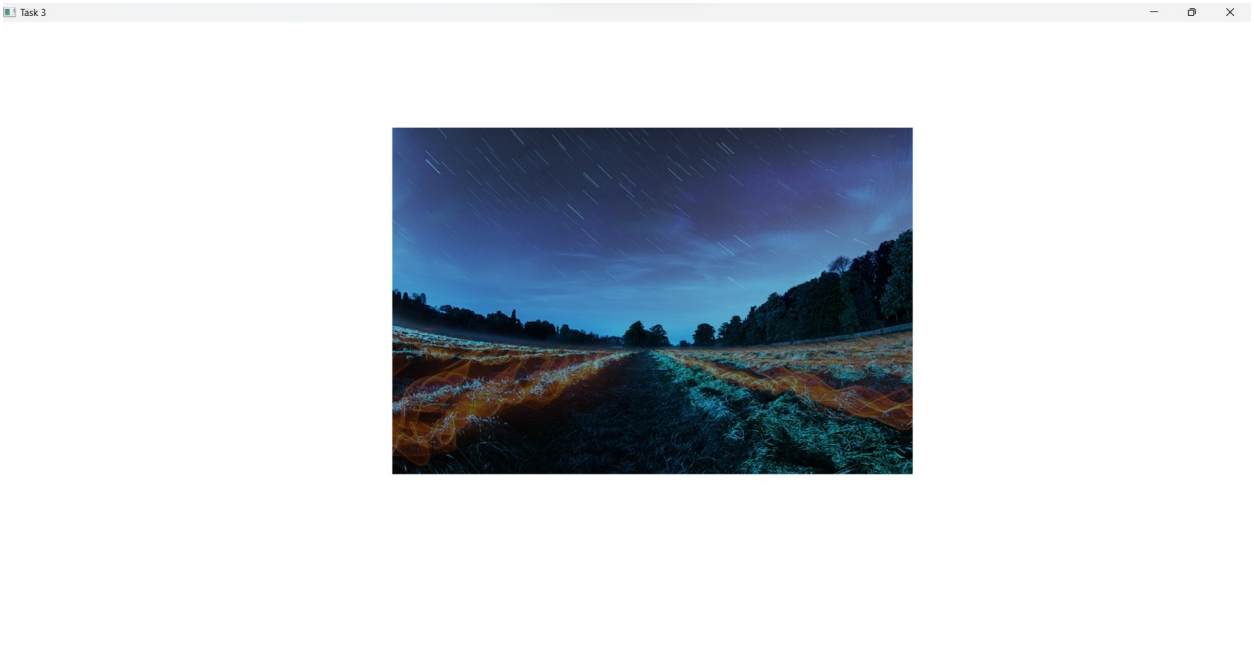

Рис. 7 Изображение, использующее случайные значения всех каналов

Вывод: в ходе лабораторной работы были изучены битовые образы, метод обработки файлов изображений с расширением tga.