МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №1

по дисциплине: Компьютерная графика
тема: «Растровые алгоритмы»

Выполнил: ст. группы ПВ-233
Мовчан Антон Юрьевич

Проверили:
ст. пр. Осипов Олег Васильевич

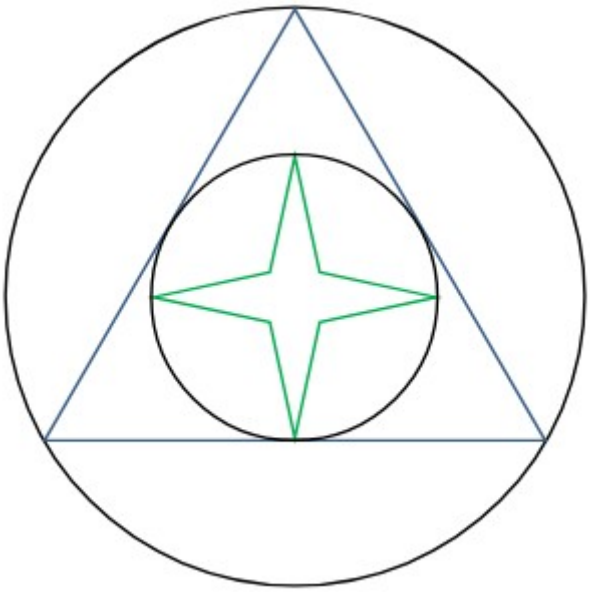
Белгород 2025 г.

Лабораторная работа №1
Вариант 8

Цель работы: изучение алгоритмов Брезенхейма растеризации графических примитивов: отрезков, окружностей.

Порядок выполнения работы

1. Изучить целочисленные алгоритмы Брезенхейма для растеризации окружности и линии.
2. Разработать алгоритм и составить программу для построения на экране изображения в соответствии с номером варианта (по журналу старосты). В качестве исходных данных взять указанные в таблице №1.



Задание: Реализовать вращение 4-конечной звезды против часовой стрелки.

Решение: Пусть W — ширина экрана, H — высота экрана. Размер рисунка a равен 7/8min(W,H). Разделим значение a пополам, тогда радиус окружности R — a / √2. Длина стороны вписанного треугольника a_t — 3 * R / √3. Радиус вписанной в треугольник окружности r_t — R / 2. Также введем центр экрана C — (W / 2, H / 2).
Вектора определяющие точки треугольника будут равны:
A = (C.x - a_t / 2, C.y + r_t)
B = (C.x, C.y - R)
C = (C.x + a_t / 2, C.y + r_t)

Для рисования звезды сделает отступ от центра, тогда r_s — r_t / 5.
Вектора определяющие точки звезды будут равны:
A = (C.x - r_t, C.y)
A1 = (C.x — r_s, C.y + r_s)
B = (C.x, C.y + r_t)

B1 = (C.x + r_s, C.y + r_s)
C = (C.x + r_t, C.y)
C1 = (C.x + r_s, C.y — r_s)
D = (C.x, C.y — r_t)
D1 = (C.x — r_s, C.y — r_s)

Для поворота звезды вокруг точки C нужно выполнить преобразование:
x` = (x — C.x) * cos(angle) — (y — C.y) * sin(angle) + C.x
y` = (x — C.x) * sin(angle) + (y — C.y) * cos(angle) + C.y

Для рисования двух окружностей, треугольника и звезды с использованием приведенных формул представлен текст на языке C++:

Painter.h

```
#ifndef PAINTER_H
#define PAINTER_H

#include "Frame.h"

// Угол поворота фигуры
float global_angle = 0;

// Координаты последнего пикселя, который выбрал пользователь
struct
{
	int X, Y;
} global_clicked_pixel = {-1, -1};

class Painter
{
public:
	void Draw(Frame &frame)
	{
		// Шахматная текстура
		for (int y = 0; y < frame.height; y++)
			for (int x = 0; x < frame.width; x++)
			{
				if ((x + y) % 2 == 0)
					frame.SetPixel(x, y, {230, 255, 230}); // Золотистый цвет
				// frame.SetPixel(x, y, { 217, 168, 14 });
				else
					frame.SetPixel(x, y, {200, 200, 200}); // Чёрный цвет
//
frame.SetPixel(x, y, { 255, 255, 255 }); // Белый цвет
			}

		int W = frame.width, H = frame.height;
		// Размер рисунка возьмём меньше (7 / 8), чтобы он не касался границ экрана
		float a = 7.0f / 8 * ((W < H) ? W - 1 : H - 1) / sqrt(2);
		if (a < 1)
			return;                                    // Если окно очень маленькое, то ничего не рисуем
		float angle = global_angle; // Угол поворота
		a = a / 2;

		// радиус окружности
		float R = int(a * sqrt(2) + 0.5f);
		// длина стороны вписанного треугольника
```

```
		float a_t = 3 * R / sqrt(3);
		// радиус вписаной в треуголтник окружности
		float r_t = a_t * sqrt(3) / 6;

		// Инициализируем исходные координаты центра и вершин треугольника
		struct
		{
			float x;
			float y;
		} C = {(float)(W / 2), (float)(H / 2)}, A[3] = {{C.x - a_t / 2, C.y + r_t}, {C.x, C.y - R}, {C.x + a_t / 2, C.y + r_t}};

		// Рисуем стороны треугольника
		for (int i = 0; i < 3; i++)
		{
			int i2 = (i + 1) % 3;
			frame.DrawLine( // Добавляем везде 0.5f, чтобы вещественные числа правильно округлялись при преобразовании к целому типу
				int(A[i].x + 0.5f),
				int(A[i].y + 0.5f),
				int(A[i2].x + 0.5f),
				int(A[i2].y + 0.5f), COLOR(0, 0, 255));
		}

		// Инициализируем исходные координаты вершин звезды
		float r_star = r_t / 5;
		struct
		{
			float x;
			float y;
		} B[8] = {{C.x - r_t, C.y}, {C.x - r_star, C.y + r_star}, {C.x, C.y + r_t}, {C.x + r_star, C.y + r_star}, {C.x + r_t, C.y}, {C.x + r_star, C.y - r_star}, {C.x, C.y - r_t}, {C.x - r_star, C.y - r_star}};

		// Поворачиваем все вершины звезды вокруг точки C на угол angle
		for (int i = 0; i < 8; i++)
		{
			float xi = B[i].x, yi = B[i].y;
			B[i].x = (xi - C.x) * cos(-angle) - (yi - C.y) * sin(-angle) + C.x;
			B[i].y = (xi - C.x) * sin(-angle) + (yi - C.y) * cos(-angle) + C.y;
		}

		// Рисуем стороны звезды
		for (int i = 0; i < 8; i++)
		{
			int i2 = (i + 1) % 8;
			frame.DrawLine( // Добавляем везде 0.5f, чтобы вещественные числа правильно округлялись при преобразовании к целому типу
				int(B[i].x + 0.5f),
				int(B[i].y + 0.5f),
				int(B[i2].x + 0.5f),
				int(B[i2].y + 0.5f), COLOR(0, 128, 0));
		}

		// длина стороны вписанного треугольника
		float a_t_2 = 3 * r_t / sqrt(3);
		// радиус вписаной в треуголтник окружности
		float r_t_2 = a_t_2 * sqrt(3) / 6;

		// Инициализируем исходные координаты вершин треугольника2
		struct
		{
			float x;
			float y;
		} D[3] = {{C.x - a_t_2 / 2, C.y + r_t_2}, {C.x, C.y - r_t}, {C.x + a_t_2 / 2, C.y + r_t_2}};
```

```
		// Поворачиваем все вершины треугольника вокруг точки C на угол angle
		for (int i = 0; i < 3; i++)
		{
			float xi = D[i].x, yi = D[i].y;
			D[i].x = (xi - C.x) * cos(angle) - (yi - C.y) * sin(angle) + C.x;
			D[i].y = (xi - C.x) * sin(angle) + (yi - C.y) * cos(angle) + C.y;
		}

		// Рисуем стороны треугольника2
		for (int i = 0; i < 3; i++)
		{
			int i2 = (i + 1) % 3;
			frame.DrawLine( // Добавляем везде 0.5f, чтобы вещественные числа правильно округлялись при преобразовании к целому типу
				int(D[i].x + 0.5f),
				int(D[i].y + 0.5f),
				int(D[i2].x + 0.5f),
				int(D[i2].y + 0.5f), COLOR(255, 0, 0));
		}

		// Рисуем описанную окружность
		frame.Circle((int)C.x, (int)C.y, int(a * sqrt(2) + 0.5f), COLOR(0, 0, 0));

		frame.Circle((int)C.x, (int)C.y, r_t + 0.5f, COLOR(0, 0, 0));

		// Рисуем пиксель, на который кликнул пользователь
		if (global_clicked_pixel.X >= 0 && global_clicked_pixel.X < W &&
			global_clicked_pixel.Y >= 0 && global_clicked_pixel.Y < H)
			frame.SetPixel(global_clicked_pixel.X, global_clicked_pixel.Y, {34, 175, 60}); // Пиксель зелёного цвета
	}
};

#endif // PAINTER_H
```

Frame.h

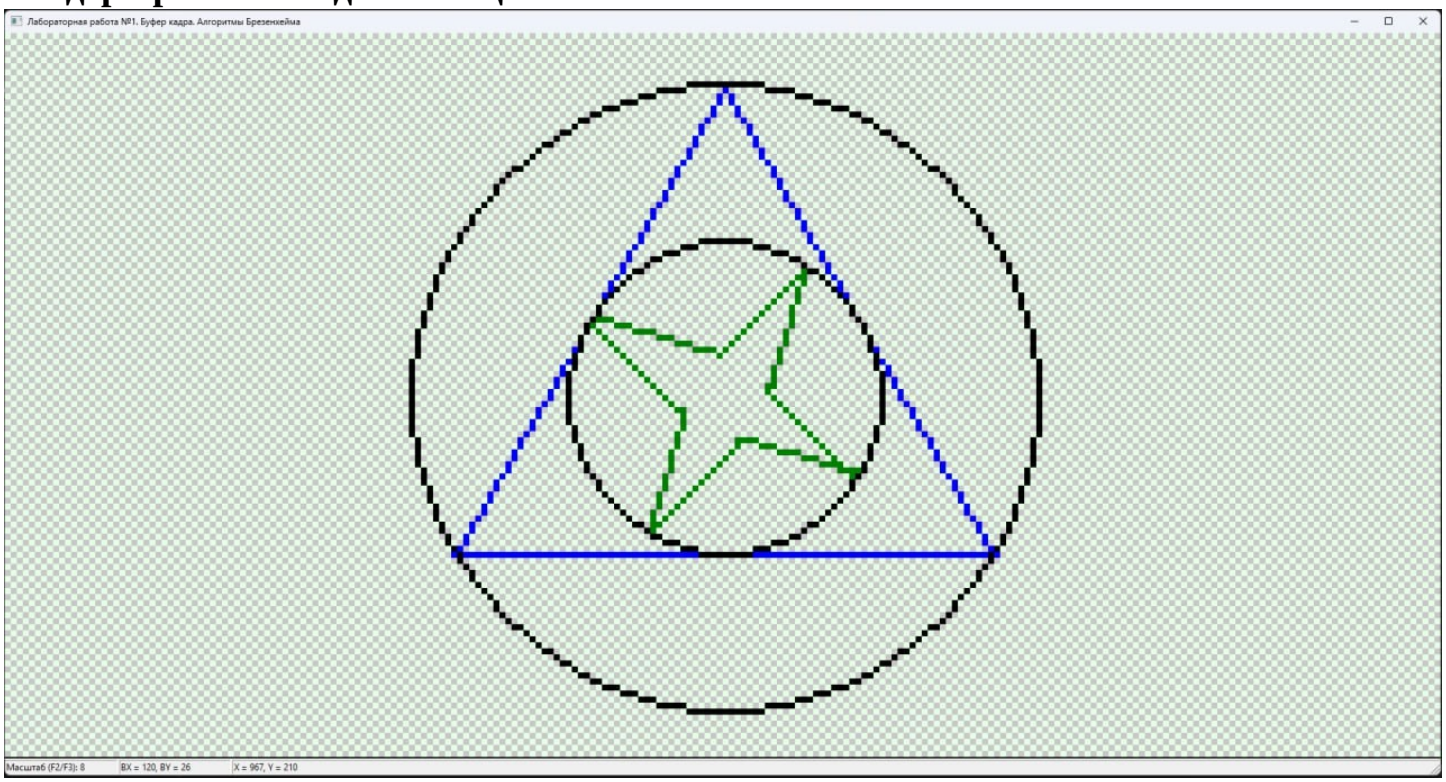
```
#ifndef FRAME_H
#define FRAME_H

#include <math.h>

// Структура для задания цвета
typedef struct tagCOLOR
{
	unsigned char RED;		// Компонента красного цвета
	unsigned char GREEN; // Компонента зелёного цвета
	unsigned char BLUE;		// Компонента синего цвета
	unsigned char ALPHA; // Прозрачность (альфа канал)

	tagCOLOR() : RED(0), GREEN(0), BLUE(0), ALPHA(255) {}
	tagCOLOR(unsigned char red, unsigned char green, unsigned char blue, unsigned char alpha = 255) : RED(red), GREEN(green), BLUE(blue), ALPHA(alpha) {}

} COLOR;

template <typename TYPE>
void swap(TYPE &a, TYPE &b)
{
	TYPE t = a;
	a = b;
	b = t;
}
```

```
// Буфер кадра
class Frame
{
	// Указатель на массив пикселей
	// Буфер кадра будет представлять собой матрицу, которая располагается в памяти в виде непрерывного блока
	COLOR *pixels;

	// Указатели на строки пикселей буфера кадра
	COLOR **matrix;

public:
	// Размеры буфера кадра
	int width, height;

	Frame(int _width, int _height) : width(_width), height(_height)
	{
		int size = width * height;

		// Создание буфера кадра в виде непрерывной матрицы пикселей
		pixels = new COLOR[size];

		// Указатели на строки пикселей запишем в отдельный массив
		matrix = new COLOR *[height];

		// Инициализация массива указателей
		for (int i = 0; i < height; i++)
		{
			matrix[i] = pixels + i * width;
		}
	}

	// Задаёт цвет color пикселю с координатами (x, y)
	void SetPixel(int x, int y, COLOR color)
	{
		matrix[y][x] = color;
	}

	// Возвращает цвет пикселя с координатами (x, y)
	COLOR GetPixel(int x, int y)
	{
		return matrix[y][x];
	}

	void Pixel8(int x0, int y0, int x, int y, COLOR color)
	{
		SetPixel(x0 + x, y0 + y, color);
		SetPixel(x0 + x, y0 - y, color);
		SetPixel(x0 + y, y0 + x, color);
		SetPixel(x0 + y, y0 - x, color);
		SetPixel(x0 - x, y0 + y, color);
		SetPixel(x0 - x, y0 - y, color);
		SetPixel(x0 - y, y0 + x, color);
		SetPixel(x0 - y, y0 - x, color);
	}

	void Circle(int x0, int y0, int radius, COLOR color)
	{
		int x = 0;
		int y = radius;
		int d = 3 - 2 * radius;

		Pixel8(x0, y0, x, y, color);
```

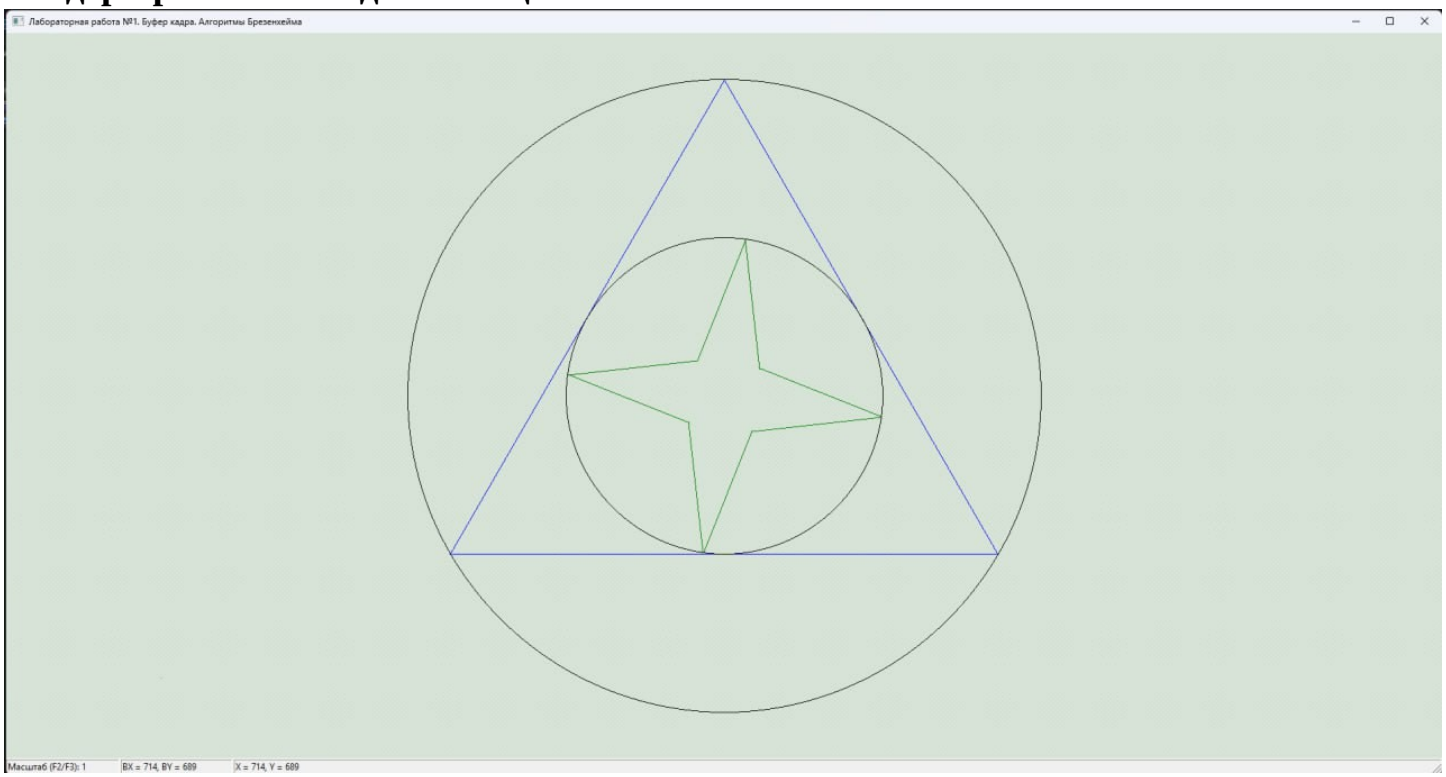
```
		while (x < y)
		{
			if (d < 0)
			{
				d = d + 4 * x + 6;
			}
			else
			{
				d = d + 4 * (x - y) + 10;
				y = y - 1;
			}
			x = x + 1;
			Pixel8(x0, y0, x, y, color);
		}
	}

	//Рисование отрезка
	void DrawLine(int x1, int y1, int x2, int y2, COLOR color)
	{
		int dy = y2 - y1, dx = x2 - x1;
		if (dx == 0 && dy == 0)
		{
			matrix[y1][x1] = color;
			return;
		}

		if (abs(dx) > abs(dy))
		{
			if (x2 < x1)
			{
				// Обмен местами точек (x1, y1) и (x2, y2)
				swap(x1, x2);
				swap(y1, y2);
				dx = -dx;
				dy = -dy;
			}

			int y = y1;
			int sign_factor = dy < 0 ? 1 : -1;
			int sumd = -2 * (y - y1) * dx + sign_factor * dx;
			for (int x = x1; x <= x2; x++)
			{
				if (sign_factor * sumd < 0)
				{
					y -= sign_factor;
					sumd += sign_factor * dx;
				}

				sumd += dy;

				matrix[y][x] = color;
			}
		}
		else
		{
			if (y2 < y1)
			{
				// Обмен местами точек (x1, y1) и (x2, y2)
				swap(x1, x2);
				swap(y1, y2);
				dx = -dx;
				dy = -dy;
			}
```

```
                        int x = x1;
                        int sign_factor = dx > 0 ? 1 : -1;
                        int sumd = 2 * (x - x1) * dy + sign_factor * dy;
                        for (int y = y1; y <= y2; y++)
                        {
                                if (sign_factor * sumd < 0)
                                {
                                        x += sign_factor;
                                        sumd += sign_factor * dy;
                                }

                                sumd -= dx;

                                matrix[y][x] = color;
                        }
                }
        }

        ~Frame(void)
        {
                delete[] pixels;
                delete[] matrix;
        }
};

#endif // FRAME_H
```

Рендер при низкой детализации:



Рендер при высокой детализации:



Вывод: В ходе выполнения лабораторной работы были изучены алгоритмы Брезенхейма растеризации графических примитивов: отрезков, окружностей.