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-223
Пахомов Владислав Андреевич

Проверили:
ст. пр. Осипов Олег Васильевич

Белгород 2024 г.

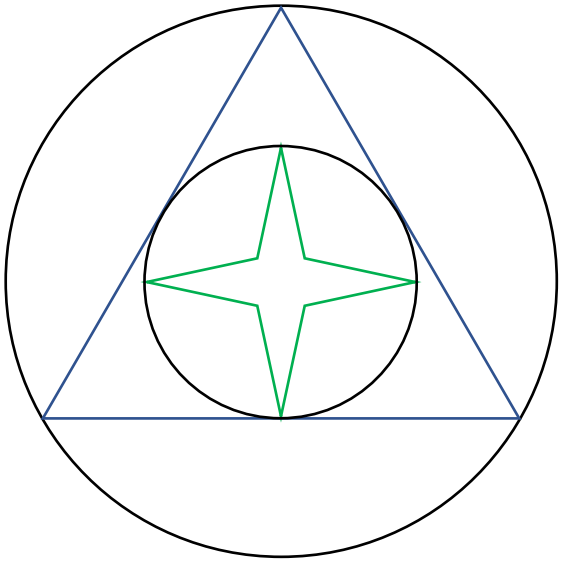
Лабораторная работа №1
Растровые алгоритмы
Вариант 8

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

Задания для выполнения к работе:

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.

Пусть W – ширина экрана, H – высота экрана. Диаметр описанной вокруг равностороннего треугольника окружности примем равным $R = 7/8 \min(W, H)$. Диаметр вписанной в треугольник окружности будет в два раза меньше описанного r = R / 2. Сторона треугольника будет равна $t = \frac{3R}{\sqrt{3}}$. Введём также центр экрана $C = (W / 2, H / 2)$. Вектор определяющие точки треугольника будут равны:

$$A_\Delta = \left(C_x, C_y - R\right)$$
$$B_\Delta = \left(C_x - \frac{t}{2}, C.y + \frac{R}{2}\right)$$
$$C_\Delta = \left(C_x + \frac{t}{2}, C_y + \frac{R}{2}\right)$$

С окружностями с радиусом R и r и центром C получим равносторонний треугольник с описанной и вписанной окружностью.

Высота и ширина четырёхконечной звезды будет равна радиусу вписанной окружности, однако пока что обозначим её центр в точке (0, 0). Также введём некоторый отступ от центра для частей, где соединяются выступы звезды. Он будет равен $S_p = \frac{r}{12}$. Тогда звезда задаётся координатами

$$A_S = (0, r), C_S = (S_P, S_P), D_S = (r, 0), E_S = (S_P, -S_P)$$
$$F_S = (0, -r), G_S = (-S_P, -S_P), H_S = (-r, 0), I_S = (-S_P, S_P)$$

Для выполнения трансформаций над звездой будем использовать SRT-матрицу. Для преобразования координат нужно переменожить каждую из них на SRT матрицу:

$$V' = M_{SRT} \cdot V$$
$$M_{SRT} = M_S \cdot M_R \cdot M_T$$

Где M_S, M_R, M_T соответственно отвечают за размер, вращение и перемещение вектора. Для двухмерного пространства достаточно будет использовать трёхмерные матрицы. Матрица M_S составляется следующим образом:

$$\mathrm{M}_S = \begin{matrix} L_W & 0 & 0 \\ 0 & L_H & 0, \\ 0 & 0 & 1 \end{matrix}$$

$L_W -$ коэф. увел. поширине, $L_H -$ коэф. увел. повысоте

$$\mathrm{M}_R = \begin{matrix} cos(\phi) & sin(\phi) & 0 \\ -sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{matrix}$$

$\phi -$ угол поворота

$$\mathrm{M}_T = \begin{matrix} 1 & 0 & X \\ 0 & 1 & Y \\ 0 & 0 & 1 \end{matrix}$$

$\mathrm{X}, \mathrm{Y} -$ смещение относительно центра

В нашем случае размер не изменяется ($L\_W = L\_H = 1$), а смещение относительно центра = C ($X = C_X; Y = C_Y$). Для рисования варианта задания представлен текст программы на C++:

Frame.h

```
// Рисование окружности
void Circle(int x0, int y0, int radius, COLOR color)
{
    int x = 0, y = radius;
    int DSUM = 2 * x * x + 2 * y * y - 2 * radius * radius - 2 * y + 1;
    while(x < y)
    {
        // Если ближе точка (x, y - 1), то смещаемся к ней
        if (DSUM > 0) {
            DSUM -= 4 * y - 4;
            y--;
        }

        // Перенос и отражение вычисленных координат на все октанты окружности
        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);
        x++;
        DSUM -= -4 * x - 2;
    }
}

// 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;
        }
    }
}
```

Matrices.h

```
#pragma once

#include <string>
#include <vector>

class Vector {
public:
    double vector[3];
    Vector(std::initializer_list<double> v) {
        memcpy(vector, v.begin(), sizeof(double) * 3);
    }
    Vector(std::vector<double> v) {
        memcpy(vector, &v[0], sizeof(double) * 3);
    }
};

class Matrix {
public:
    double data[9];
    double *matrix[3];
    Matrix(std::initializer_list<double> v) {
        memcpy(data, v.begin(), sizeof(double) * 9);
        matrix[0] = data;
        matrix[1] = data + 3;
        matrix[2] = data + 6;
    }
    Matrix(std::vector<double> v) {
        memcpy(data, &v[0], sizeof(double) * 9);
        matrix[0] = data;
        matrix[1] = data + 3;
        matrix[2] = data + 6;
    }

    Matrix multiply(Matrix& another) {
        double dataNew[9] = {};
        double* matrixNew[3];
        matrixNew[0] = dataNew;
        matrixNew[1] = dataNew + 3;
        matrixNew[2] = dataNew + 6;
        for (int i = 0; i < 3; i++) {
```

```
            for (int j = 0; j < 3; j++) {
                matrixNew[i][j] = 0;

                for (int k = 0; k < 3; k++) {
                    matrixNew[i][j] += this->matrix[i][k] * another.matrix[k][j];
                }
            }
        }

        return Matrix(std::vector<double>(dataNew, dataNew + 9));
    }

    Vector multiply(Vector& vec) {
        return Vector({
            vec.vector[0] * this->matrix[0][0] + vec.vector[1] * this->matrix[0][1] +
vec.vector[2] * this->matrix[0][2],
            vec.vector[0] * this->matrix[1][0] + vec.vector[1] * this->matrix[1][1] +
vec.vector[2] * this->matrix[1][2],
            vec.vector[0] * this->matrix[2][0] + vec.vector[1] * this->matrix[2][1] +
vec.vector[2] * this->matrix[2][2] });
    }
};
```

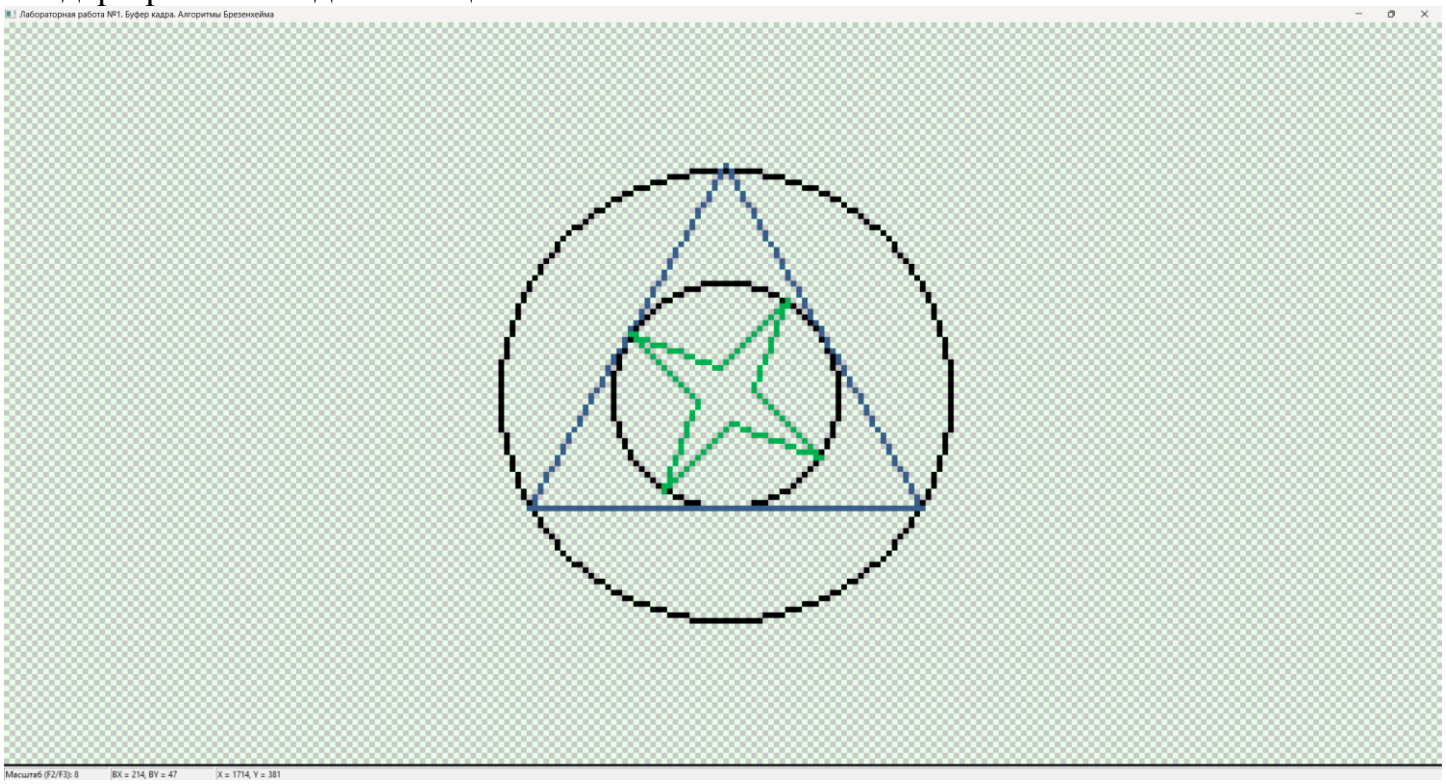
Painter.h

```
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;
        coordinate C = { W / 2, H / 2 };

        // Рисуем описанную окружность
        frame.Circle((int)C.x, (int)C.y, (int)a, COLOR(0, 0, 0));
        // Рисуем вписанную окружность
        frame.Circle((int)C.x, (int)C.y, (int)(a * 0.5), COLOR(0, 0, 0));
        //Рисуем треугольник
        double t = (3 * a) / sqrt(3);
        coordinate triangleA = { C.x, C.y - a };
        coordinate triangleB = { C.x - t / 2, C.y + a / 2 };
        coordinate triangleC = { C.x + t / 2, C.y + a / 2 };
        frame.DrawLine(triangleA.x + 0.5, triangleA.y + 0.5, triangleB.x + 0.5, triangleB.y +
0.5, { 56, 93, 138 });
        frame.DrawLine(triangleC.x + 0.5, triangleC.y + 0.5, triangleB.x + 0.5, triangleB.y +
0.5, { 56, 93, 138 });
        frame.DrawLine(triangleA.x + 0.5, triangleA.y + 0.5, triangleC.x + 0.5, triangleC.y +
0.5, { 56, 93, 138 });

        Matrix S = { 1, 0, 0,
                     0, 1, 0,
                     0, 0, 1 };
        Matrix R = { cos(angle), -sin(angle),  0,
                     sin(angle),  cos(angle),  0,
                              0,           0,  1 };
        Matrix T = { 1, 0, W / 2.0,
                     0, 1, H / 2.0,
                     0, 0,       1 };
        Matrix SRT = (T.multiply(R)).multiply(S);
        double starOffset = a / 12;
        coordinate star[8] = {
            { 0, a / 2 },
            { starOffset, starOffset },
            { a / 2, 0 },
            { starOffset, -starOffset },
            { 0, -a / 2 },
            { -starOffset, -starOffset },
```

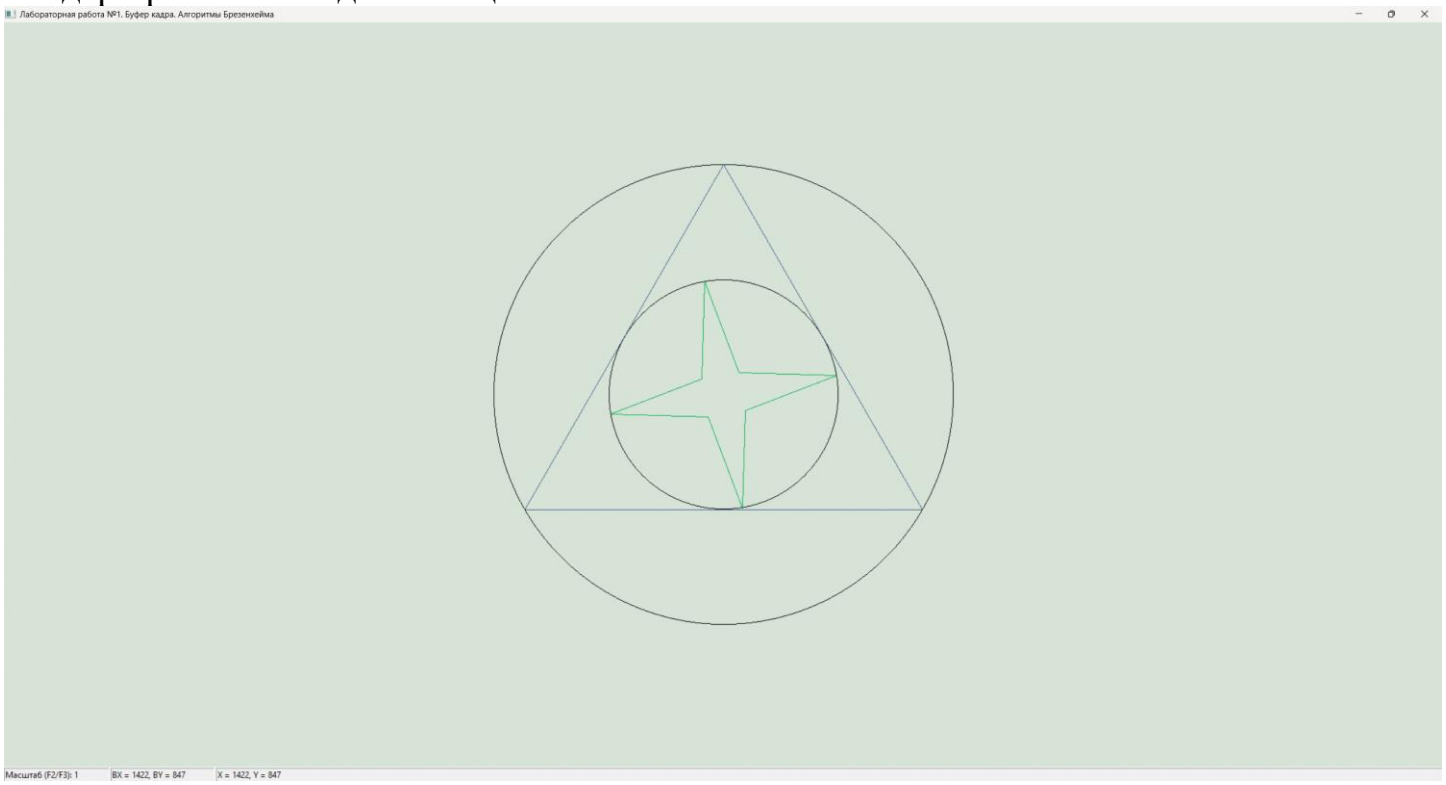
```
            { -a / 2, 0 },
            { -starOffset, starOffset } };

        for (int i = 0; i < 8; i++)
        {
            Vector pointVector = { star[i].x, star[i].y, 1 };
            pointVector = SRT.multiply(pointVector);
            star[i].x = pointVector.vector[0];
            star[i].y = pointVector.vector[1];
        }

        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(star[i].x + 0.5f),
                int(star[i].y + 0.5f),
                int(star[i2].x + 0.5f),
                int(star[i2].y + 0.5f), COLOR(0, 176, 80));
        }

        // 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
```

Ссылка на репозиторий:
https://github.com/IAmProgrammist/comp_graphics/tree/main/lab_1_basics

Рендер при низкой детализации



Рендер при высокой детализации



Вывод: в ходе лабораторной работы получены навыки создания простейших ассемблерных программ с использованием пакета masm32, получены навыки пользования отладчиком x32dbg.