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)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №1

по дисциплине: Компьютерная графика

тема: «Растровые алгоритмы»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

Осипов Олег Васильевич

Белгород 2024 г.

# Содержание:

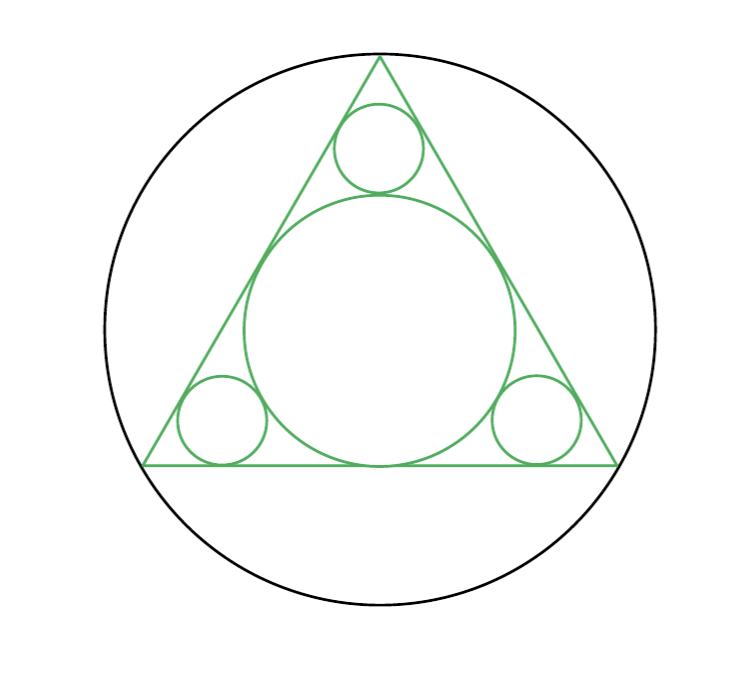
1. Название темы.
2. Цель работы.
3. Постановка задачи.
4. Вывод необходимых геометрических формул для построения изображения.
5. Реализации алгоритмов Брезенхейма для рисования отрезка и окружности.
6. Текст программы для рисования основных фигур.
7. Результат работы программы (снимки экрана).
8. Вывод о проделанной работе.

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

**Задачи**:

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

Вариант 3:



**Задание:** Реализовать вращение внутреннего зелёного треугольника против часовой стрелки.

1. **Описанная окружность**

Описанная окружность — это окружность, которая проходит через все вершины треугольника. Её центр совпадает с центром треугольника, а радиус равен расстоянию от центра до любой из вершин.

Формулы для описанной окружности:

Центр окружности: C()— центр окна.

Радиус окружности: R = , где W и H — ширина и высота окна.

Этот радиус масштабирует окружность так, чтобы она не касалась границ окна. Умножение на делает окружность чуть меньше.

**Алгоритм:**

1. Вычисляем центр окружности
2. Радиус окружности равен R =
3. Вызываем метод frame.Circle() для отрисовки окружности с этим центром и радиусом.

Код:

// Радиус описанной окружности

float radius = 7.0f / 8 \* ((W < H) ? W - 1 : H - 1) / 2;

// Рисуем описанную окружность

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

**2. Вершины треугольника**

Треугольник вписан в описанную окружность. Его вершины лежат на окружности и равномерно распределены по углам: 0°, 120°, 240°. Треугольник вращается на угол θ, который изменяется во времени.

**Формулы для вершин треугольника:**

1. Первая вершина:
2. Вторая вершина​ (поворот на 120°):
3. Третья вершина​ (поворот на 240°):

Здесь θ — это угол вращения треугольника.

**Алгоритм:**

1. Определяем радиус описанной окружности R.
2. Вычисляем координаты трёх вершин треугольника по углам 0°, 120°, 240° и с учётом текущего угла вращения.
3. Соединяем вершины треугольника с помощью метода frame.DrawLine().

**Код:**

// Вершины треугольника на окружности с радиусом `radius` и углами 0, 120 и 240 градусов

struct

{

float x;

float y;

} A[3] = {

{ C.x + radius \* cos(angle), C.y + radius \* sin(angle) }, // Первая вершина (угол 0)

{ C.x + radius \* cos(angle + 2.0f \* M\_PI / 3.0f), C.y + radius \* sin(angle + 2.0f \* M\_PI / 3.0f) }, // Вторая вершина (угол 120 градусов)

{ C.x + radius \* cos(angle + 4.0f \* M\_PI / 3.0f), C.y + radius \* sin(angle + 4.0f \* M\_PI / 3.0f) } // Третья вершина (угол 240 градусов)

};

**3. Вписанная окружность**

**Вписанная окружность** — это окружность, которая касается всех сторон треугольника. Её центр является центром треугольника по отношению к его вершинам, а радиус можно вычислить через площадь треугольника и полупериметр.

**Формулы для вписанной окружности:**

1. Полупериметр треугольника:, где a, b, c — длины сторон треугольника.
2. Площадь треугольника (по формуле Герона):
3. Радиус вписанной окружности:
4. Центр вписанной окружности:,

**Алгоритм:**

1. Вычисляем длины сторон треугольника a, b, c.
2. Вычисляем полупериметр p и площадь S.
3. Радиус вписанной окружности r находим по формуле ​.
4. Находим центр вписанной окружности I(x,y).
5. Вращаем центр вписанной окружности на угол вращения треугольника, чтобы окружность вращалась синхронно с треугольником.
6. Рисуем окружность методом frame.Circle().

**Код:**

// Вычисляем центр вписанной окружности (Ix, Iy)

float Ix = (a \* A[0].x + b \* A[1].x + c \* A[2].x) / (a + b + c);

float Iy = (a \* A[0].y + b \* A[1].y + c \* A[2].y) / (a + b + c);

// Вращаем центр вписанной окружности вокруг центра треугольника (точка C) на угол angle

float rotated\_Ix = (Ix - C.x) \* cos(angle) - (Iy - C.y) \* sin(angle) + C.x;

float rotated\_Iy = (Ix - C.x) \* sin(angle) + (Iy - C.y) \* cos(angle) + C.y;

// Рисуем вписанную окружность зелёного цвета

frame.Circle((int)(rotated\_Ix + 0.5f), (int)(rotated\_Iy + 0.5f), int(r\_incircle + 0.5f), COLOR(34, 139, 34)); // Вписанная окружность

**4. Три дополнительные окружности**

Эти окружности касаются двух сторон треугольника и вписанной окружности. Для их построения мы выбираем точку, находящуюся между вершинами треугольника и центром вписанной окружности.

**Алгоритм:**

1. Для каждой вершины треугольника ​ находим направление к центру вписанной окружности.
2. Вычисляем новую точку, сдвигаясь по направлению к вершине от центра вписанной окружности на расстояние, равное радиусу вписанной окружности плюс некоторое фиксированное смещение.
3. Радиус этих окружностей выбираем фиксированным.
4. Рисуем каждую окружность методом frame.Circle().

**Код:**

// Внешние окружности между углами треугольника и вписанной окружностью

for (int i = 0; i < 3; i++)

{

// Смещаем центр этой окружности ближе к вершине

float direction\_x = (A[i].x - rotated\_Ix);

float direction\_y = (A[i].y - rotated\_Iy);

float length = sqrt(direction\_x \* direction\_x + direction\_y \* direction\_y);

direction\_x /= length; // Нормализуем вектор

direction\_y /= length;

// Центр внешней окружности на расстоянии от вписанной окружности

float distance\_to\_incircle = r\_incircle + (radius \* 0.165f); // Устанавливаем расстояние от вписанной окружности

// Центр внешней окружности

float outer\_circle\_x = rotated\_Ix + direction\_x \* distance\_to\_incircle;

float outer\_circle\_y = rotated\_Iy + direction\_y \* distance\_to\_incircle;

// Радиус внешней окружности: 1/4 радиуса описанной окружности

float outer\_radius = radius \* 0.165f; // Динамически устанавливаем радиус внешней окружности

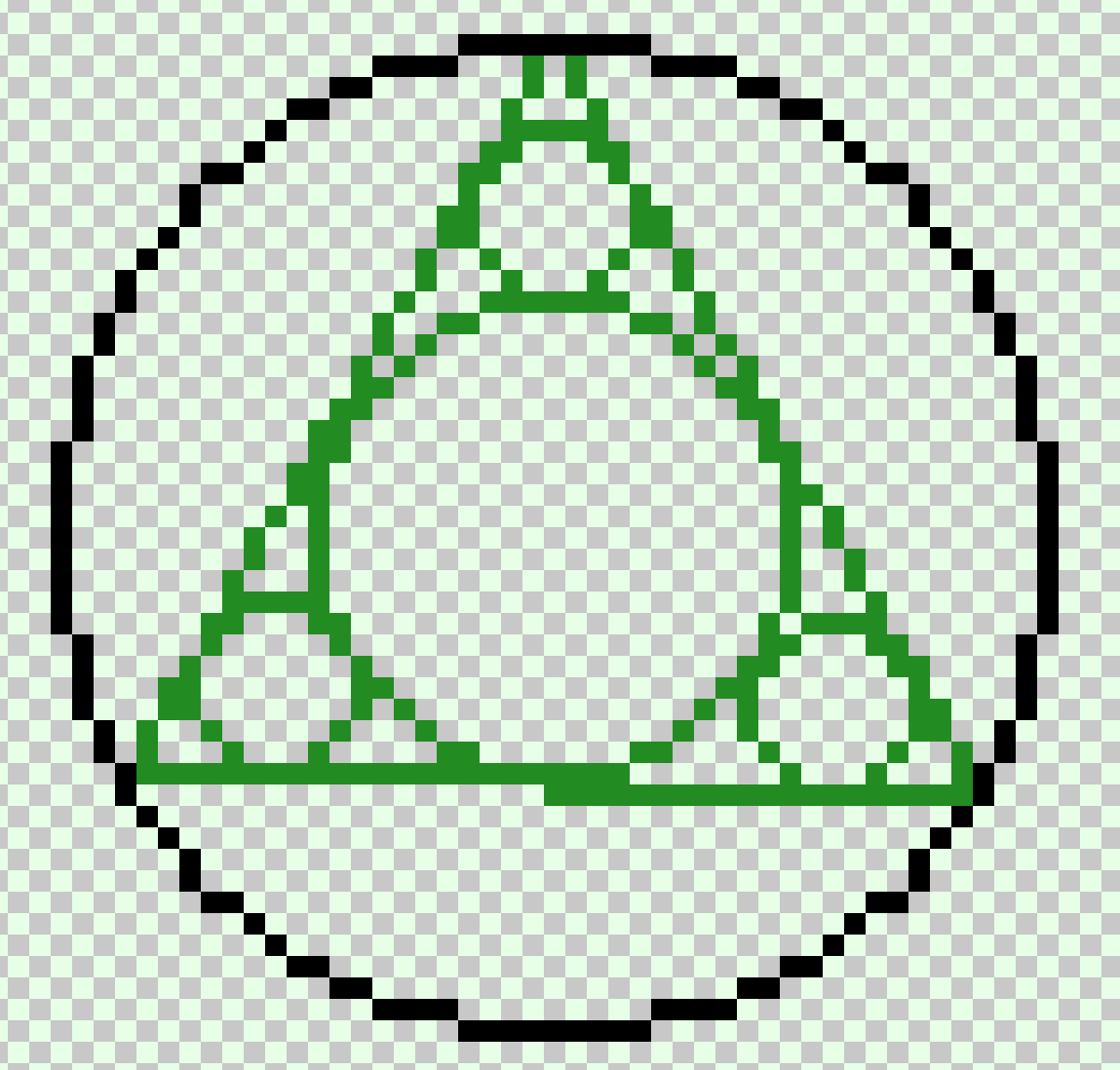
// Рисуем дополнительные окружности зелёного цвета

frame.Circle((int)(outer\_circle\_x + 0.5f), (int)(outer\_circle\_y + 0.5f), int(outer\_radius + 0.5f), COLOR(34, 139, 34));

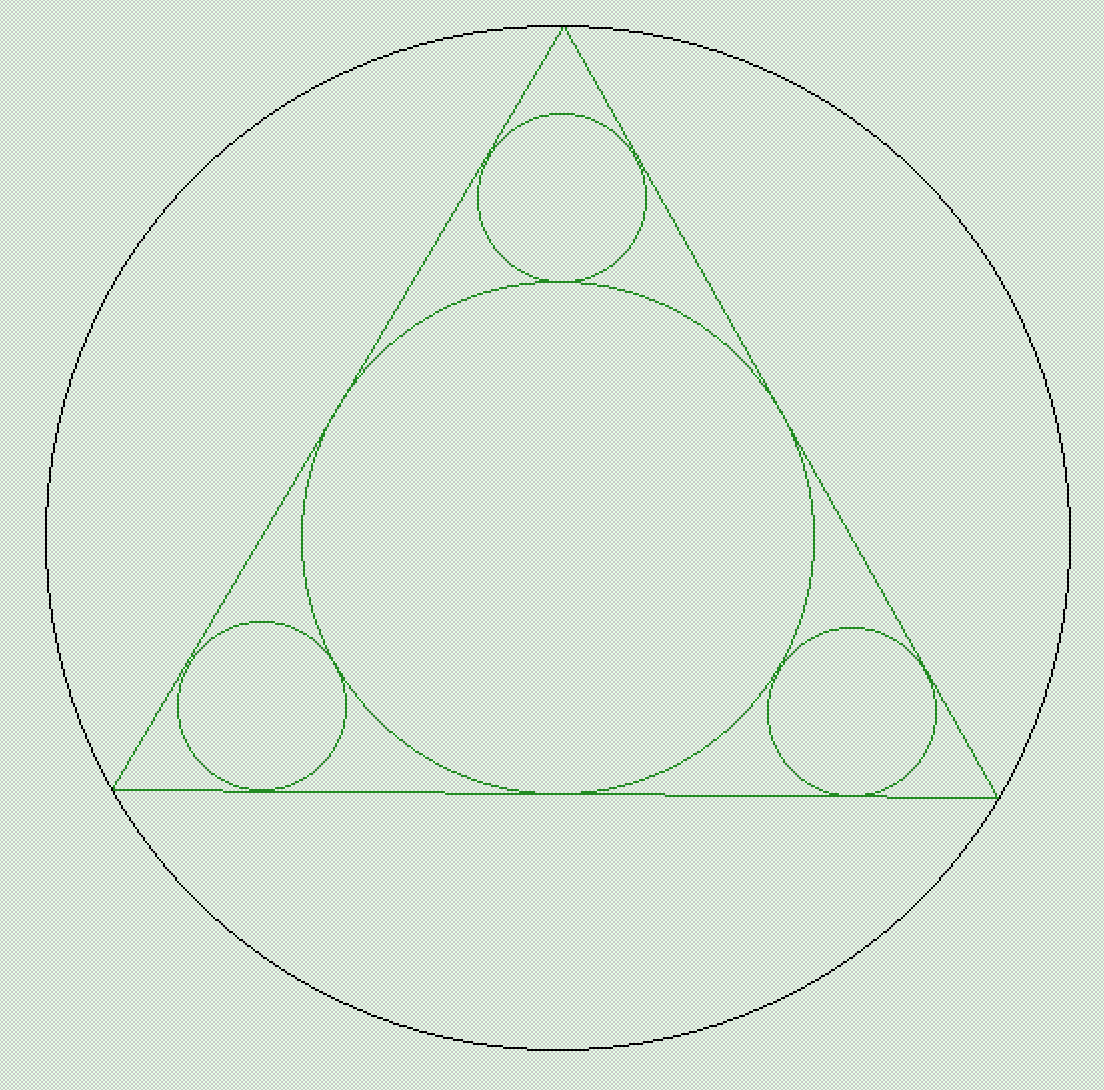
}

Таким образом, данный алгоритм последовательно вычисляет и рисует все окружности и треугольник с учётом их вращения.

Внешний вид фигур при низком разрешении:



Внешний вид фигур при высоком разрешении:

****

При низком и высоком разрешении дефектов не обнаружено.

Код программы:

Frame.h

#ifndef FRAME\_H

#define FRAME\_H

#include <math.h>

// Cтруктура 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];

}

// Рисование окружности

void Circle(int x0, int y0, int radius, COLOR color)

{

int x = 0, y = radius;

while(x < y)

{

// Определяем, какая точка (пиксель): (x, y) или (x, y - 1) ближе к линии окружности

int D1 = x \* x + y \* y - radius \* radius;

int D2 = x \* x + (y - 1) \* (y - 1) - radius \* radius;

// Если ближе точка (x, y - 1), то смещаемся к ней

if (D1 > -D2)

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++;

}

}

// 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, dx2 = dx / 2, p = 0;

if (dy < 0) dx2 = -dx2;

for (int x = x1; x <= x2; x++)

{

// y = (dy \* (x - x1) + dx2) / dx + y1;

y = (p + dx2) / dx + y1;

p += 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, dy2 = dy / 2, p = 0;

if (dx < 0) dy2 = -dy2;

for (int y = y1; y <= y2; y++)

{

// x = (dx \* (y - y1) + dy2) / dy + x1;

x = (p + dy2) / dy + x1;

p += dx;

matrix[y][x] = color;

}

}

}

~Frame(void)

{

delete []pixels;

delete []matrix;

}

};

#endif // FRAME\_H

Painter.h

#ifndef PAINTER\_H

#define PAINTER\_H

#include "Frame.h"

// Определение числа π, если оно не определено

#ifndef M\_PI

#define M\_PI 3.14159265358979323846

#endif

// 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

else

frame.SetPixel(x, y, { 200, 200, 200 }); // Серый цвет

}

int W = frame.width, H = frame.height;

// Радиус описанной окружности

float radius = 7.0f / 8 \* ((W < H) ? W - 1 : H - 1) / 2;

if (radius < 1) return; // Если окно очень маленькое, то ничего не рисуем

float angle = global\_angle; // Угол поворота

// Центр описанной окружности (центра треугольника)

struct

{

float x;

float y;

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

// Вершины треугольника на окружности с радиусом `radius` и углами 0, 120 и 240 градусов

struct

{

float x;

float y;

} A[3] = {

{ C.x + radius \* cos(angle), C.y + radius \* sin(angle) }, // Первая вершина (угол 0)

{ C.x + radius \* cos(angle + 2.0f \* M\_PI / 3.0f), C.y + radius \* sin(angle + 2.0f \* M\_PI / 3.0f) }, // Вторая вершина (угол 120 градусов)

{ C.x + radius \* cos(angle + 4.0f \* M\_PI / 3.0f), C.y + radius \* sin(angle + 4.0f \* M\_PI / 3.0f) } // Третья вершина (угол 240 градусов)

};

// Рисуем стороны треугольника

for (int i = 0; i < 3; i++)

{

int i2 = (i + 1) % 3;

frame.DrawLine(

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(34, 139, 34)); // Зелёный цвет

}

// Вычисляем длины сторон треугольника

float a = sqrt(pow(A[1].x - A[2].x, 2) + pow(A[1].y - A[2].y, 2)); // Сторона между вершинами 1 и 2

float b = sqrt(pow(A[0].x - A[2].x, 2) + pow(A[0].y - A[2].y, 2)); // Сторона между вершинами 0 и 2

float c = sqrt(pow(A[0].x - A[1].x, 2) + pow(A[0].y - A[1].y, 2)); // Сторона между вершинами 0 и 1

// Полупериметр

float p = (a + b + c) / 2;

// Площадь треугольника по формуле Герона

float area = sqrt(p \* (p - a) \* (p - b) \* (p - c));

// Радиус вписанной окружности

float r\_incircle = area / p;

// Вычисляем центр вписанной окружности (Ix, Iy)

float Ix = (a \* A[0].x + b \* A[1].x + c \* A[2].x) / (a + b + c);

float Iy = (a \* A[0].y + b \* A[1].y + c \* A[2].y) / (a + b + c);

// Вращаем центр вписанной окружности вокруг центра треугольника (точка C) на угол angle

float rotated\_Ix = (Ix - C.x) \* cos(angle) - (Iy - C.y) \* sin(angle) + C.x;

float rotated\_Iy = (Ix - C.x) \* sin(angle) + (Iy - C.y) \* cos(angle) + C.y;

// Рисуем вписанную окружность зелёного цвета

frame.Circle((int)(rotated\_Ix + 0.5f), (int)(rotated\_Iy + 0.5f), int(r\_incircle + 0.5f), COLOR(34, 139, 34)); // Вписанная окружность

// Внешние окружности между углами треугольника и вписанной окружностью

for (int i = 0; i < 3; i++)

{

// Смещаем центр этой окружности ближе к вершине

float direction\_x = (A[i].x - rotated\_Ix);

float direction\_y = (A[i].y - rotated\_Iy);

float length = sqrt(direction\_x \* direction\_x + direction\_y \* direction\_y);

direction\_x /= length; // Нормализуем вектор

direction\_y /= length;

// Центр внешней окружности на расстоянии от вписанной окружности

float distance\_to\_incircle = r\_incircle + (radius \* 0.165f); // Устанавливаем расстояние от вписанной окружности

// Центр внешней окружности

float outer\_circle\_x = rotated\_Ix + direction\_x \* distance\_to\_incircle;

float outer\_circle\_y = rotated\_Iy + direction\_y \* distance\_to\_incircle;

// Радиус внешней окружности: 1/4 радиуса описанной окружности

float outer\_radius = radius \* 0.165f; // Динамически устанавливаем радиус внешней окружности

// Рисуем дополнительные окружности зелёного цвета

frame.Circle((int)(outer\_circle\_x + 0.5f), (int)(outer\_circle\_y + 0.5f), int(outer\_radius + 0.5f), COLOR(34, 139, 34));

}

// Рисуем описанную окружность

frame.Circle((int)C.x, (int)C.y, int(radius + 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

**Вывод:** в ходе работы изучены алгоритмы Брезенхейма растеризации графических примитивов: отрезков, окружностей. С помощью отрезков были построены и анимированы многоугольники. Собраны в изображение требуемое по варианту.