

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-233

Мороз Роман Алексеевич

Проверил:

Осипов Олег Васильевич

Белгород 2025

Вариант 9

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

Радиус внешней окружности: $R = (\min(W, H) / 2) \times \frac{7}{8}$

Радиус внутренней окружности: $r = R / 3$

Положение основания треугольников: $R_base = r \times 1.5$

Вершина на внешней окружности

$$x_1 = centerX + R \times \cos(\theta + i \times \pi/4)$$

$$y_1 = centerY + R \times \sin(\theta + i \times \pi/4)$$

Вершины основания (ближе к центру)

$$x_2 = centerX + R_base \times \cos(\theta + i \times \pi/4 - \alpha)$$

$$y_2 = centerY + R_base \times \sin(\theta + i \times \pi/4 - \alpha)$$

$$x_3 = centerX + R_base \times \cos(\theta + i \times \pi/4 + \alpha)$$

$$y_3 = centerY + R_base \times \sin(\theta + i \times \pi/4 + \alpha)$$

Для линии от (x_1, y_1) до (x_2, y_2) :

$$\Delta x = |x_2 - x_1|$$

$$\Delta y = |y_2 - y_1|$$

$$sx = \text{sign}(x_2 - x_1)$$

$$sy = \text{sign}(y_2 - y_1)$$

Масштабирование в viewport:

Размер viewport: $V = (7 \times \min(W_window, H_window)) / 8$

Смещение viewport:

$$V_x = (W_window - V) / 2$$

$$V_y = (H_window - V) / 2$$

Преобразование логических координат в физические:

$$x_{physical} = V_x + (x_{logical} / W_{logical}) \times V$$

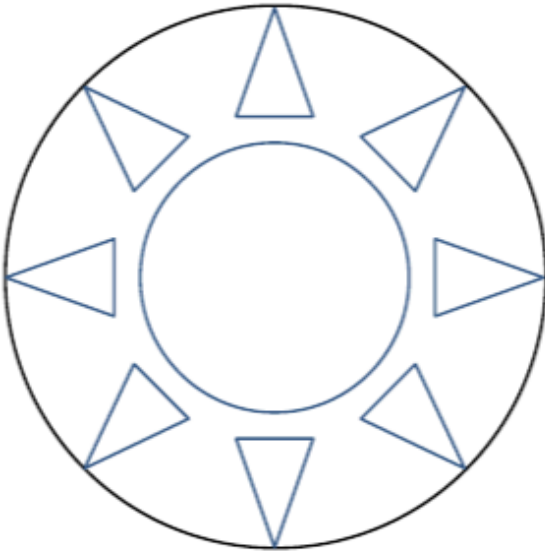
$$y_{physical} = V_y + (y_{logical} / H_{logical}) \times V$$

Угловая скорость и положение:

Угловая скорость: $\omega = 0.02 \text{ рад/кадр}$

Угол поворота: $\theta(t) = \theta_0 + \omega \times t$

Период полного оборота: $T = 2\pi / \omega = 314 \text{ кадров}$

9,18,27		Реализовать вращение треугольников вокруг центрального круга.
---------	---	---

```
#include <SDL2/SDL.h>
#include <cmath>
#include <vector>
#include <algorithm>
#include <string>

/**
 * @brief Константы для начального размера окна
 */
const int INITIAL_WIDTH = 800;
const int INITIAL_HEIGHT = 600;

/**
```

```

* @brief Константы для ограничения разрешения
*/
const int MIN_RESOLUTION = 10;    // Минимальное разрешение (10x10 пикселей)
const int MAX_RESOLUTION = 2000;  // Максимальное разрешение (2000x2000
пикселей)

/**
* @brief Структура для представления цвета в формате RGBA
*/
struct COLOR {
    Uint8 r; // Красная компонента (0-255)
    Uint8 g; // Зеленая компонента (0-255)
    Uint8 b; // Синяя компонента (0-255)
    Uint8 a; // Альфа-канал (прозрачность, 0-255)
};

/**
* @brief Класс для работы с буфером кадра
*
* Представляет собой матрицу пикселей, в которой хранится изображение
* перед отрисовкой на экран. Это предотвращает мерцание при отрисовке.
*/
class Frame {
    int width;           // Ширина буфера кадра
    int height;          // Высота буфера кадра
    std::vector<COLOR> pixels; // Вектор пикселей (width * height элементов)

public:
    /**
    * @brief Конструктор буфера кадра
    * @param w Ширина буфера
    * @param h Высота буфера
    */
    Frame(int w, int h) : width(w), height(h), pixels(w * h) {}

    /**
    * @brief Изменяет размер буфера кадра
    * @param w Новая ширина буфера
    * @param h Новая высота буфера
    */
    void Resize(int w, int h) {
        width = w;
        height = h;
        pixels.resize(w * h);
    }

    /**
    * @brief Устанавливает цвет пикселя с указанными координатами
    * @param x Координата X пикселя
    * @param y Координата Y пикселя
    * @param color Цвет для установки

```

```

    */
void SetPixel(int x, int y, COLOR color) {
    if (x >= 0 && x < width && y >= 0 && y < height) {
        pixels[y * width + x] = color;
    }
}

/**
 * @brief Возвращает цвет пикселя с указанными координатами
 * @param x Координата X пикселя
 * @param y Координата Y пикселя
 * @return Цвет пикселя или черный цвет, если координаты вне диапазона
 */
COLOR GetPixel(int x, int y) const {
    if (x >= 0 && x < width && y >= 0 && y < height) {
        return pixels[y * width + x];
    }
    return {0, 0, 0, 0};
}

/**
 * @brief Очищает буфер кадра указанным цветом
 * @param color Цвет для очистки
 */
void Clear(COLOR color) {
    std::fill(pixels.begin(), pixels.end(), color);
}

/**
 * @brief Возвращает ширину буфера кадра
 * @return Ширина буфера кадра
 */
int GetWidth() const { return width; }

/**
 * @brief Возвращает высоту буфера кадра
 * @return Высота буфера кадра
 */
int GetHeight() const { return height; }
};

/**
 * @brief Оптимизированный алгоритм Брезенхейма для рисования линии
 * @param frame Буфер кадра для рисования
 * @param x1, y1 Координаты начальной точки линии
 * @param x2, y2 Координаты конечной точки линии
 * @param color Цвет линии
 */
void DrawLine(Frame& frame, int x1, int y1, int x2, int y2, COLOR color) {
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);

```

```

int sx = (x1 < x2) ? 1 : -1;
int sy = (y1 < y2) ? 1 : -1;
int err = dx - dy;
int err2;

while (true) {
    frame.SetPixel(x1, y1, color);
    if (x1 == x2 && y1 == y2) break;

    err2 = 2 * err;
    if (err2 > -dy) {
        err -= dy;
        x1 += sx;
    }
    if (err2 < dx) {
        err += dx;
        y1 += sy;
    }
}
}

/**
 * @brief Оптимизированный алгоритм Брезенхейма для рисования окружности
 * @param frame Буфер кадра для рисования
 * @param x0, y0 Координаты центра окружности
 * @param radius Радиус окружности
 * @param color Цвет окружности
 */
void DrawCircle(Frame& frame, int x0, int y0, int radius, COLOR color) {
    if (radius <= 0) return;

    int x = 0;
    int y = radius;
    int d = 3 - 2 * radius;

    // Вспомогательная функция для отрисовки точек окружности
    auto drawCirclePoints = [&](int x, int y) {
        frame.SetPixel(x0 + x, y0 + y, color);
        frame.SetPixel(x0 - x, y0 + y, color);
        frame.SetPixel(x0 + x, y0 - y, color);
        frame.SetPixel(x0 - x, y0 - y, color);
        frame.SetPixel(x0 + y, y0 + x, color);
        frame.SetPixel(x0 - y, y0 + x, color);
        frame.SetPixel(x0 + y, y0 - x, color);
        frame.SetPixel(x0 - y, y0 - x, color);
    };

    while (y >= x) {
        drawCirclePoints(x, y);

        x++;

```

[illegible]

```

if (!renderer) {
    SDL_Log("Не удалось создать рендерер: %s", SDL_GetError());
    SDL_DestroyWindow(window);
    SDL_Quit();
    return 1;
}

// Получение формата пикселей
SDL_PixelFormat* format = SDL_AllocFormat(SDL_PIXELFORMAT_RGBA32);
if (!format) {
    SDL_Log("Не удалось получить формат пикселей: %s", SDL_GetError());
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    return 1;
}

// Инициализация буфера кадра
int logicalWidth = 500;
int logicalHeight = 500;
Frame frame(logicalWidth, logicalHeight);

// Создание текстуры
SDL_Texture* texture = SDL_CreateTexture(renderer,
                                           SDL_PIXELFORMAT_RGBA32,
                                           SDL_TEXTUREACCESS_STREAMING,
                                           logicalWidth, logicalHeight);

if (!texture) {
    SDL_Log("Не удалось создать текстуру: %s", SDL_GetError());
    SDL_FreeFormat(format);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    return 1;
}

// Основные переменные
bool running = true;
SDL_Event event;
double globalRotation = 0.0;
const double rotationSpeed = 0.02; // Скорость вращения (радианов за кадр)

// Функция для обновления заголовка окна с информацией о разрешении
auto updateWindowTitle = [window, &logicalWidth, &logicalHeight]() {
    std::string title = "Вращение треугольников - Разрешение: " +
                        std::to_string(logicalWidth) + "x" +
                        std::to_string(logicalHeight);
    SDL_SetWindowTitle(window, title.c_str());
};

// Установка начального заголовка

```



```

updateWindowTitle();

// Основной цикл программы
while (running) {
    Uint32 frameStart = SDL_GetTicks();

    // Обработка событий
    while (SDL_PollEvent(&event)) {
        if (event.type == SDL_QUIT) {
            running = false;
        } else if (event.type == SDL_KEYDOWN) {
            if (event.key.keysym.sym == SDLK_F2) {
                // Уменьшение логического разрешения с шагом 10 пикселей
                logicalWidth = std::max(MIN_RESOLUTION, logicalWidth - 10);
                logicalHeight = logicalWidth;
                frame.Resize(logicalWidth, logicalHeight);
                SDL_DestroyTexture(texture);
                texture = SDL_CreateTexture(renderer,
                                            SDL_PIXELFORMAT_RGBA32,
                                            SDL_TEXTUREACCESS_STREAMING,
                                            logicalWidth, logicalHeight);

                updateWindowTitle();
            } else if (event.key.keysym.sym == SDLK_F3) {
                // Увеличение логического разрешения с шагом 10 пикселей
                logicalWidth = std::min(MAX_RESOLUTION, logicalWidth + 10);
                logicalHeight = logicalWidth;
                frame.Resize(logicalWidth, logicalHeight);
                SDL_DestroyTexture(texture);
                texture = SDL_CreateTexture(renderer,
                                            SDL_PIXELFORMAT_RGBA32,
                                            SDL_TEXTUREACCESS_STREAMING,
                                            logicalWidth, logicalHeight);

                updateWindowTitle();
            }
        }
    }

    // Обновление угла вращения
    globalRotation += rotationSpeed;
    if (globalRotation > 2 * M_PI) {
        globalRotation -= 2 * M_PI;
    }

    // Очистка буфера кадра (белый фон)
    frame.Clear({255, 255, 255, 255});

    // Определение центра и радиуса
    int centerX = logicalWidth / 2;
    int centerY = logicalHeight / 2;
    int circleRadius = (std::min(logicalWidth, logicalHeight) / 2) * 7 / 8;

```

```

// Рисование внешнего круга (черная граница)
DrawCircle(frame, centerX, centerY, circleRadius, {0, 0, 0, 255});

// Рисование центрального круга (темно-синий)
COLOR darkBlue = {0, 0, 139, 255};
int innerCircleRadius = circleRadius / 3;
DrawCircle(frame, centerX, centerY, innerCircleRadius, darkBlue);

// Рисование треугольников (темно-синие)
int triangleCount = 8;
int triangleBaseDistance = innerCircleRadius * 1.5; // Основание ближе к
внутреннему кругу

for (int i = 0; i < triangleCount; i++) {
    double angle = i * (2 * M_PI / triangleCount) + globalRotation;

    // Вершина треугольника на внешнем круге
    int x1 = centerX + static_cast<int>(circleRadius * cos(angle));
    int y1 = centerY + static_cast<int>(circleRadius * sin(angle));

    // Основание треугольника (ближе к внутреннему кругу)
    int x2 = centerX + static_cast<int>(triangleBaseDistance * cos(angle
- 0.2));
    int y2 = centerY + static_cast<int>(triangleBaseDistance * sin(angle
- 0.2));

    int x3 = centerX + static_cast<int>(triangleBaseDistance * cos(angle
+ 0.2));
    int y3 = centerY + static_cast<int>(triangleBaseDistance * sin(angle
+ 0.2));

    // Рисование треугольника
    DrawTriangle(frame, x1, y1, x2, y2, x3, y3, darkBlue);
}

// Обновление текстуры из буфера кадра
void* texturePixels;
int pitch;
SDL_LockTexture(texture, NULL, &texturePixels, &pitch);
Uint32* texPixels = static_cast<Uint32*>(texturePixels);

// Копирование данных из буфера кадра в текстуру
for (int y = 0; y < logicalHeight; y++) {
    for (int x = 0; x < logicalWidth; x++) {
        COLOR color = frame.GetPixel(x, y);
        texPixels[y * (pitch / sizeof(Uint32)) + x] =
            SDL_MapRGBA(format, color.r, color.g, color.b, color.a);
    }
}
SDL_UnlockTexture(texture);

```

```
// Очистка рендерера
SDL_RenderClear(renderer);

// Установка вьюпорта (7/8 от минимального размера окна по центру)
int windowHeight, windowHeight;
SDL_GetWindowSize(window, &windowWidth, &windowHeight);
int viewportSize = (7 * std::min(windowWidth, windowHeight)) / 8;
int viewportX = (windowWidth - viewportSize) / 2;
int viewportY = (windowHeight - viewportSize) / 2;
SDL_Rect viewport = {viewportX, viewportY, viewportSize, viewportSize};
SDL_RenderSetViewport(renderer, &viewport);

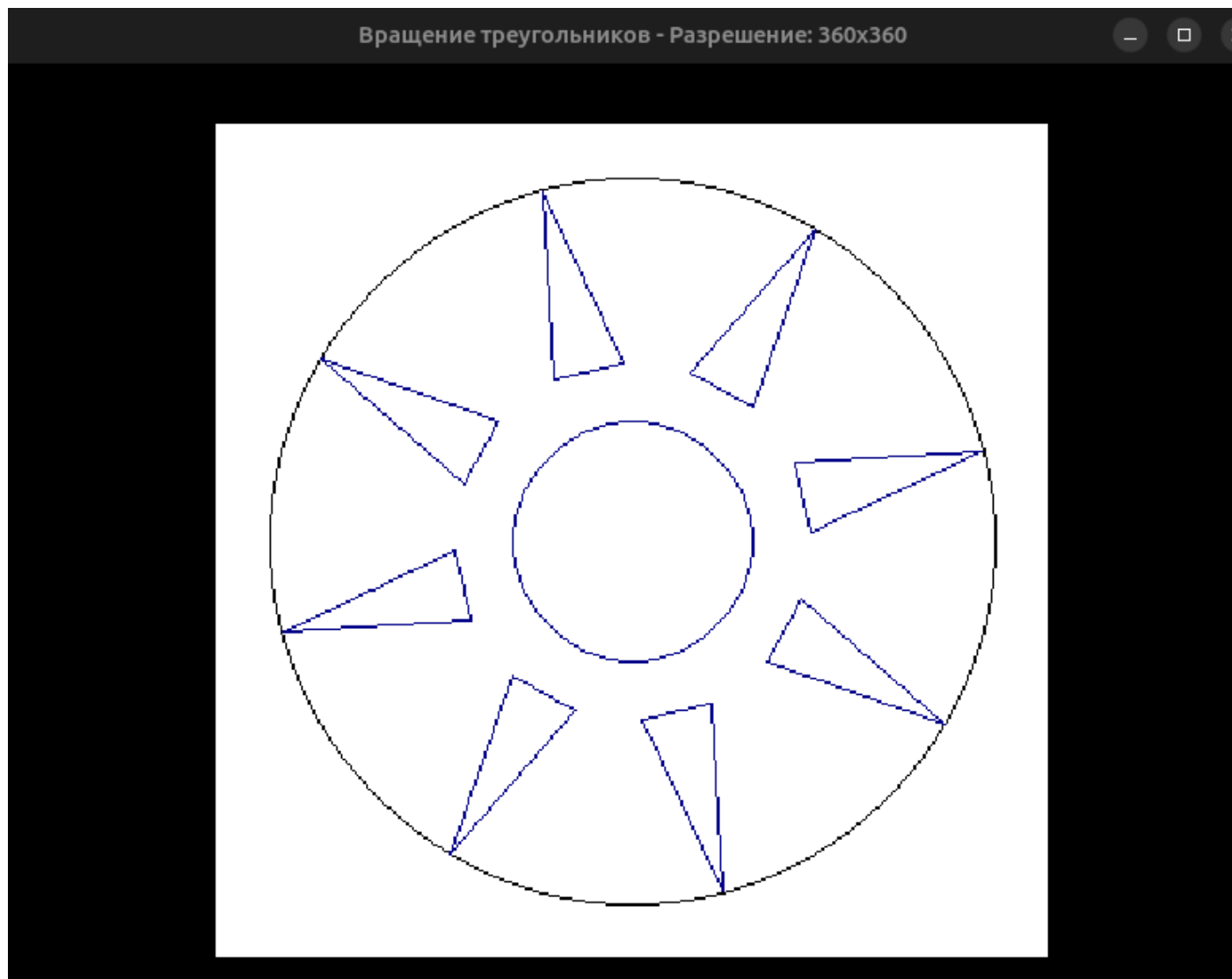
// Копирование текстуры на рендерер
SDL_RenderCopy(renderer, texture, NULL, NULL);

// Отображение рендерера
SDL_RenderPresent(renderer);

// Ограничение частоты кадров (30 FPS)
Uint32 frameTime = SDL_GetTicks() - frameStart;
if (frameTime < 33) {
    SDL_Delay(33 - frameTime);
}

// Освобождение ресурсов
SDL_FreeFormat(format);
SDL_DestroyTexture(texture);
SDL_DestroyRenderer(renderer);
SDL_DestroyWindow(window);
SDL_Quit();

return 0;
}
```



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