Implementación de Algoritmos Gráficos Básicos

Julio Enrique Viche Castillo

16 de junio de 2025

Repositorio del Proyecto en GitHub

1. Introducción

Este trabajo presenta la implementación de algoritmos gráficos clásicos (DDA, Bresenham y FlowFill) en una aplicación Windows Forms moderna. El sistema desarrollado combina la precisión matemática de estos algoritmos con características contemporáneas como programación asíncrona y una interfaz gráfica interactiva, permitiendo a los usuarios visualizar y controlar el proceso de trazado de líneas, círculos y relleno de áreas.

2. Descripción de la Interfaz

2.1. Componentes Principales

La interfaz gráfica se compone de:

- **Canvas**: PictureBox de 400x400 píxeles
- **Panel de Control**: Agrupado por secciones

2.2. Secciones de Control

2.2.1. Dibujo de Líneas

- Campos de entrada: (x_0, y_0) y (x_1, y_1)
- Botón Draw DDA
- Botón Draw Bresenham

2.2.2. Dibujo de Círculos

- Campos de entrada: centro (cx, cy) y radio (r)
- Botón Circle Bresenham

2.2.3. Dibujo de Rombo

- Campos de entrada: ancho y alto
- Botón Draw Rhombus

2.3. Controles Generales

- Checkbox Animations Enabled
- Botón Cancel Animation
- Botón Clear Canvas

2.4. Herramienta de Relleno

- Activación mediante click en el canvas
- Estado controlado por variable fillEnabled
- Color de relleno predefinido (Negro)

2.5. Estados de la Interfaz

2.5.1. Estado Normal

- Todos los botones habilitados
- Relleno disponible
- Canvas receptivo a clicks

2.5.2. Estado de Animación

- Botones de dibujo deshabilitados
- Solo Cancel Animation habilitado
- Relleno temporalmente desactivado

3. Algoritmos Implementados

3.1. Algoritmo DDA (Digital Differential Analyzer)

```
public async Task DrawDDA
(
    PictureBox canvas,
    Bitmap bitmap,
    bool animationEnabled,
    CancellationToken token
)
{
    AnimationManager am = new AnimationManager
    (
        "DDA LINE",
        animationEnabled,
        canvas,
        bitmap
    );

    int steps = Math.Max(Math.Abs(dx), Math.Abs(dy));
    float xInc = (float)dx / steps;
    float yInc = (float)dy / steps;

    float xk = start.X;
    float yk = start.Y;

    am.AlgorithmStart();

    for (int i = 0; i <= steps; i++)
    {
        if (token.IsCancellationRequested)
        {
            Console.WriteLine("Algorithm cancelled.");
            return;
        }

        await am.SetPixel
```

```
            (
                (int)Math.Round(xk),
                (int)Math.Round(yk),
                Color.Black,
                2
            );

            xk += xInc;
            yk += yInc;
        }

        canvas.Invalidate();
        am.AlgorithmEnd();
    }
```

3.2. Análisis del Algoritmo DDA

3.2.1. Implementación Base

El algoritmo implementa un trazador de líneas asíncrono con los siguientes componentes:

```
steps = Math.Max(Math.Abs(dx), Math.Abs(dy))
xInc = dx / steps
yInc = dy / steps
```

3.2.2. Componentes Principales

Gestión de Animación

- Utiliza AnimationManager para control de dibujo
- Parámetros:
  - Nombre del algoritmo: "DDA LINE"
  - Estado de animación
  - Canvas y bitmap para renderizado

Cálculo de Incrementos

$$xInc = \frac{dx}{steps}, \quad yInc = \frac{dy}{steps} \tag{1}$$

3.2.3. Proceso Iterativo

Para cada paso i de 0 a *steps*:

1. Redondeo de coordenadas: (x_k, y_k)
2. Dibujo de píxel mediante SetPixel
3. Incremento de posición: $x_k += xInc$, $y_k += yInc$

3.2.4. Características Técnicas

Eficiencia

- **Complejidad Temporal**: $O(steps)$
- **Complejidad Espacial**: $O(1)$
- **Operaciones por iteración**:
  - 2 sumas en punto flotante
  - 2 operaciones de redondeo
  - 1 operación de dibujo asíncrona

Características Asíncronas

- Implementación mediante async/await
- Control de cancelación vía CancellationToken
- Actualización visual mediante Invalidate()

3.2.5. Aspectos de Implementación

- Uso de punto flotante para precisión
- Sistema de cancelación integrado
- Gestión de animaciones
- Actualización visual del canvas

3.3. Algoritmo de Bresenham para Líneas

```
public async Task DrawBresenham
(
    PictureBox canvas,
    Bitmap bitmap,
    bool animationEnabled,
    CancellationToken token
)
{
    AnimationManager am = new AnimationManager
    (
        "BRESENHAM LINE",
        animationEnabled,
        canvas,
        bitmap
    );

    int x0 = start.X;
    int y0 = start.Y;
    int x1 = end.X;
    int y1 = end.Y;

    bool steep = Math.Abs(y1 - y0) > Math.Abs(x1 - x0);

    if (steep)
    {
        (x0, y0) = (y0, x0);
        (x1, y1) = (y1, x1);
    }

    if (x0 > x1)
    {
        (x0, x1) = (x1, x0);
        (y0, y1) = (y1, y0);
    }

    int dx = x1 - x0;
```

```
    int dy = Math.Abs(y1 - y0);
    int error = dx / 2;
    int ystep = y0 < y1 ? 1 : -1;
    int y = y0;

    am.AlgorithmStart();

    for (int x = x0; x <= x1; x++)
    {
        if (token.IsCancellationRequested)
        {
            Console.WriteLine("Algorithm cancelled.");
            return;
        }

        int drawX = steep ? y : x;
        int drawY = steep ? x : y;

        await am.SetPixel
        (
            drawX,
            drawY,
            Color.Black,
            2
        );

        error -= dy;
        if (error < 0)
        {
            y += ystep;
            error += dx;
        }
    }

    canvas.Invalidate();
    am.AlgorithmEnd();
}
```

3.4. Análisis del Algoritmo de Bresenham

3.4.1. Implementación Base

El algoritmo implementa un trazador de líneas asíncrono optimizado:

Preprocesamiento

- Detección de pendiente pronunciada (steep):

$$steep = |y_1 - y_0| > |x_1 - x_0| \tag{2}$$

- Intercambio de coordenadas si es necesario
- Garantiza dirección x positiva

3.4.2. Componentes Principales

Variables de Control

- $dx = x_1 - x_0$
- $dy = |y_1 - y_0|$
- $error = dx/2$ (error inicial)
- $ystep = \pm 1$ (dirección del incremento en y)

Gestión de Animación

- AnimationManager con identificador "BRESENHAM LINE"
- Control de estado y visualización

3.4.3. Proceso Iterativo

Para cada x de x_0 a x_1:

1. Determinar coordenadas de dibujo según steep
2. Dibujar píxel en $(drawX, drawY)$
3. Actualizar error: $error- = dy$

4. Si $error < 0$:
    - Incrementar y: $y+ = ystep$
    - Corregir error: $error+ = dx$

3.4.4. Características Técnicas

Eficiencia

- **Complejidad Temporal**: $O(dx)$
- **Complejidad Espacial**: $O(1)$
- **Operaciones por iteración**:
    - 1-2 operaciones enteras
    - 1 comparación
    - 1 operación de dibujo asíncrona

Características Asíncronas

- Implementación mediante async/await
- Soporte para cancelación
- Actualización visual del canvas

3.4.5. Optimizaciones

- Uso exclusivo de aritmética entera
- Manejo de casos especiales (steep)
- Normalización de dirección x
- Error inicial optimizado ($dx/2$)

3.5. Algoritmo de Bresenham para Círculos

```
public async Task DrawBresenham
(
    PictureBox canvas,
    Bitmap bitmap,
    bool animationEnabled,
    CancellationToken token
)
{
    AnimationManager am = new AnimationManager
    (
        "BRESENHAM CIRCLE",
        animationEnabled,
        canvas,
        bitmap
    );

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

    am.AlgorithmStart();

    while (x <= y)
    {
        if (token.IsCancellationRequested)
        {
            Console.WriteLine("Algorithm cancelled.");
            return;
        }

        int xc = center.X;
        int yc = center.Y;

        await am.SetPixel(xc + x, yc + y, Color.Black, 1);
        await am.SetPixel(xc - x, yc + y, Color.Black, 0);
        await am.SetPixel(xc + x, yc - y, Color.Black, 0);
```

```
            await am.SetPixel(xc - x, yc - y, Color.Black, 0);
            await am.SetPixel(xc + y, yc + x, Color.Black, 0);
            await am.SetPixel(xc - y, yc + x, Color.Black, 0);
            await am.SetPixel(xc + y, yc - x, Color.Black, 0);
            await am.SetPixel(xc - y, yc - x, Color.Black, 0);

            if (d < 0)
            {
                d += 4 * x + 6;
            }
            else
            {
                d += 4 * (x - y) + 10;
                y--;
            }
            x++;
        }

        canvas.Invalidate();
        am.AlgorithmEnd();
    }
```

3.6. Análisis del Algoritmo de Bresenham para Círculos

3.6.1. Implementación Base

El algoritmo implementa un trazador de círculos asíncrono utilizando simetría de octantes:

Inicialización

- Variables de control iniciales:

$$x = 0$$
$$y = radio$$
$$d = 3 - 2 * radio \text{ (discriminante)}$$

3.6.2. Componentes Principales

Gestión de Animación

- AnimationManager con identificador "BRESENHAM CIRCLE"
- Control asíncrono del dibujado

Simetría de Octantes Dibuja 8 píxeles simétricos para cada punto calculado:

- $(x_c \pm x, y_c \pm y)$
- $(x_c \pm y, y_c \pm x)$

3.6.3. Proceso Iterativo

Mientras $x \leq y$:

1. Dibujar 8 píxeles simétricos
2. Actualizar discriminante d:
   - Si $d < 0$: $d+ = 4x + 6$
   - Si $d \geq 0$: $d+ = 4(x - y) + 10$ y $y - -$
3. Incrementar x

3.6.4. Características Técnicas

Eficiencia

- **Complejidad Temporal**: $O(r)$ donde r es el radio
- **Complejidad Espacial**: $O(1)$
- **Operaciones por iteración**:
  - 8 operaciones de dibujo asíncronas
  - 2-3 operaciones aritméticas enteras
  - 1 comparación

Características Asíncronas

- Implementación mediante async/await
- Soporte para cancelación
- Actualización visual del canvas

3.6.5. Optimizaciones

- Uso exclusivo de aritmética entera
- Aprovechamiento de simetría octantal
- Minimización de cálculos por iteración

3.7. Algoritmo de Relleno (FloodFill)

```
public static async Task FlowFill
(
    PictureBox canvas,
    Bitmap bitmap,
    Point start,
    Color replacementColor,
    bool animationEnabled,
    CancellationToken token
)
{
    if
    (
        start.X < 0 || start.X >= bitmap.Width ||
        start.Y < 0 || start.Y >= bitmap.Height
    )
        return;

    AnimationManager am = new AnimationManager
    (
        "FLOWFILL ALGORITHM",
        animationEnabled,
        canvas,
        bitmap
    );

    Color targetColor = bitmap.GetPixel(start.X, start.Y);

    if (targetColor.ToArgb() == replacementColor.ToArgb())
        return;
```

```
HashSet<Point> visited = new HashSet<Point>();
Stack<Point> pixels = new Stack<Point>();
pixels.Push(start);

int count = 0;

am.AlgorithmStart();

while (pixels.Count > 0)
{
    if (token.IsCancellationRequested)
    {
        Console.WriteLine("Algorithm cancelled.");
        return;
    }

    Point p = pixels.Pop();

    if
    (
        p.X < 0 || p.X >= bitmap.Width ||
        p.Y < 0 || p.Y >= bitmap.Height
    )
        continue;

    if (visited.Contains(p))
        continue;

    Color currentColor = bitmap.GetPixel(p.X, p.Y);
    if (currentColor.ToArgb() != targetColor.ToArgb())
        continue;

    visited.Add(p);

    await am.SetPixel
    (
        p.X,
```

```
                p.Y,
                replacementColor,
                count++ % 100 == 0 ? 1 : 0
            );

            pixels.Push(new Point(p.X - 1, p.Y));
            pixels.Push(new Point(p.X, p.Y + 1));
            pixels.Push(new Point(p.X + 1, p.Y));
            pixels.Push(new Point(p.X, p.Y - 1));
        }

        canvas.Invalidate();
        am.AlgorithmEnd();
    }
```

3.8. Análisis del Algoritmo FlowFill

3.8.1. Implementación Base

Implementa un algoritmo de relleno por inundación asíncrono con las siguientes estructuras:

- HashSet¡Point¿para píxeles visitados
- Stack¡Point¿para píxeles pendientes
- Validación de límites del bitmap

3.8.2. Componentes Principales

Validaciones Iniciales

- Verificación de coordenadas dentro del bitmap
- Comprobación de color objetivo vs color de reemplazo

Estructuras de Datos

$$\begin{cases} visited : \text{conjunto de píxeles procesados} \\ pixels : \text{pila de píxeles por procesar} \end{cases} \tag{3}$$

3.8.3. Proceso Iterativo

Mientras existan píxeles en la pila:

1. Extraer punto p de la pila
2. Validar límites y estado del píxel
3. Si es válido:
    - Marcar como visitado
    - Cambiar color
    - Agregar vecinos (4-conectividad):
        - $(x-1, y)$ - izquierda
        - $(x, y+1)$ - abajo
        - $(x+1, y)$ - derecha
        - $(x, y-1)$ - arriba

3.8.4. Características Técnicas

Eficiencia

- **Complejidad Temporal**: $O(w \times h)$ donde w,h son dimensiones del bitmap
- **Complejidad Espacial**: $O(w \times h)$ para estructuras auxiliares
- **Operaciones por píxel**:
    - 1 operación de GetPixel
    - 1 operación de SetPixel asíncrona
    - 4 inserciones en la pila
    - 1 inserción en conjunto visitado

Características Asíncronas

- Control de animación cada 100 píxeles
- Soporte para cancelación
- Actualización visual del canvas

3.8.5. Optimizaciones

- Uso de HashSet para búsqueda O(1)
- Validación temprana de límites
- Actualización visual optimizada
- Control de redundancia de píxeles

3.8.6. Consideraciones de Rendimiento

- Consumo de memoria proporcional al área a rellenar
- Puede requerir optimización para áreas extensas
- La implementación iterativa es más estable que la recursiva

4. Manual de Usuario

4.1. Interfaz Principal

La aplicación presenta una interfaz gráfica con:

- Canvas de dibujo (400x400 píxeles)
- Panel de controles para diferentes algoritmos
- Opciones de animación y limpieza

4.2. Dibujo de Líneas

4.2.1. Entrada de Datos

- Coordenadas iniciales (x_0, y_0)
- Coordenadas finales (x_1, y_1)
- Rango válido: [0,399] para cada coordenada

4.2.2. Algoritmos Disponibles

1. **DDA**: Botón "Draw DDA"
2. **Bresenham**: Botón "Draw Bresenham"

4.3. Dibujo de Círculos

4.3.1. Parámetros Requeridos

- Centro (x, y): Coordenadas en rango [0,399]
- Radio (r): Valor positivo
- Validación: El círculo debe estar dentro del canvas

4.4. Dibujo de Rombo

4.4.1. Parámetros

- Ancho: Valor positivo menor a 400
- Alto: Valor positivo menor a 400
- Centro: Automáticamente en el centro del canvas

4.5. Herramienta de Relleno

- Activación: Click en área a rellenar
- Color de relleno: Negro
- Disponible cuando no hay animaciones en proceso

4.6. Control de Animaciones

- **Checkbox Animations Enabled**: Activa/desactiva animaciones
- **Botón Cancel Animation**: Detiene la animación actual
- **Botón Clear Canvas**: Limpia el área de dibujo

4.7. Estados del Sistema

4.7.1. Durante la Animación

- Botones de dibujo deshabilitados
- Relleno deshabilitado
- Botón de cancelación habilitado

4.7.2. Estado Normal

- Todos los botones de dibujo habilitados
- Relleno habilitado
- Botón de cancelación deshabilitado

4.8. Mensajes de Error

El sistema muestra alertas cuando:

- Valores fuera de rango [0,399]
- Radio del círculo menor o igual a cero
- Círculo fuera de los límites del canvas
- Dimensiones de rombo inválidas