

**UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE**



**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN**

**Computación Gráfica**

**Actividad 2-2P: Algoritmos Clásicos de Computación Gráfica**

**Autores:**

Eduardo Antonio Mortensen Franco

NRC: 27873  
**Ecuador 2025-11-27**

## **1.1. Algoritmo: Dibujo de Circunferencias**

### **1.1.1. Descripción técnica general**

Para el trazado de circunferencias en una interfaz gráfica se implementaron tres métodos: Bresenham, Punto Medio y Paramétrico. Todos permiten generar puntos sobre la circunferencia utilizando diferentes estrategias matemáticas. El objetivo del estudiante es demostrar la correcta ejecución, comparar el rendimiento y analizar los resultados visuales producidos por cada variante.

Cada algoritmo trabaja directamente sobre un objeto Graphics y utiliza una rutina unificada denominada PintarPixel(), que coloca un píxel ampliado mediante un rectángulo del tamaño definido por el usuario.

El centro de la circunferencia se establece automáticamente en el centro del área de dibujo (PictureBox) y el usuario proporciona el radio y el tamaño del píxel mediante controles numéricos.

## **1.2. Variantes implementadas**

### **1.2.1. Variante 1: Bresenham**

#### **a) Descripción técnica**

Este método utiliza una lógica incremental basada en aritmética entera para decidir, en cada iteración, si el punto de la circunferencia debe desplazarse horizontalmente o diagonalmente.

El algoritmo aprovecha la simetría de los ocho octantes, permitiendo calcular ocho puntos en cada paso.

La variable de decisión  $p$  determina el ajuste en el eje y cuando la aproximación al borde cambia de signo.

#### **b) Justificación**

Es una versión clásica y eficiente porque utiliza únicamente sumas y restas. Garantiza buena velocidad y precisión para radios medianos y grandes. Además, al no utilizar funciones trigonométricas, su desempeño es superior al método paramétrico.

#### **c) Caso de uso demostrado**

El estudiante demuestra la generación uniforme de los ocho octantes utilizando únicamente operaciones enteras. Se verifican circunferencias con radios pequeños, medianos y grandes para evaluar la suavidad del borde.

### **1.2.2. Variante 2: Punto Medio (Midpoint Circle)**

#### **a) Descripción técnica**

Utiliza la función implícita del círculo y evalúa la posición relativa del punto medio entre dos posibles incrementos.

Si el punto medio está fuera de la circunferencia, se ajusta descendiendo en el eje y. Si se encuentra dentro, solo se incrementa en el eje x.

También utiliza la propiedad de simetría para dibujar los ocho octantes.

#### **b) Justificación**

Ofrece resultados equivalentes a Bresenham pero con un razonamiento matemático más simple. Es un método apropiado para enseñanza porque permite visualizar claramente la relación entre el criterio geométrico y la generación de puntos.

#### **c) Caso de uso demostrado**

El estudiante verifica la correcta modificación del término de decisión d, mostrando resultados idénticos a Bresenham pero con diferente lógica de cálculo.

### **1.2.3. Variante 3: Método Paramétrico**

#### **a) Descripción técnica**

Evalúa las coordenadas mediante las funciones trigonométricas:

$$x = cx + r \cos(\theta)$$

$$y = cy + r \sin(\theta)$$

Se recorren valores de  $\theta$  desde 0 hasta  $2\pi$ , y el incremento depende del tamaño del radio para evitar huecos en el trazo.

#### **b) Justificación**

Es útil para visualizaciones continuas, círculos suaves o con antialiasing. Aunque es menos eficiente debido al uso de funciones trigonométricas, permite mayor control sobre el estilo del trazo y permite experimentar con pasos de ángulo personalizados.

#### **c) Caso de uso demostrado**

El estudiante muestra cómo el cambio en el paso angular afecta la densidad de puntos y la suavidad del borde, verificando el impacto del rendimiento frente a las variantes basadas en aritmética entera.

### **1.3. Instrucciones de uso del formulario (FormCirculo)**

1. Seleccionar el radio de la circunferencia mediante numericRadio.
2. Seleccionar el tamaño del píxel (ampliación visual) mediante numericPixel.
3. Elegir un color mediante el botón btnColor.
4. Seleccionar el algoritmo presionando uno de los siguientes botones:
  - Dibujar Bresenham
  - Dibujar Punto Medio
  - Dibujar Paramétrico
5. El área de dibujo se limpia automáticamente antes de trazar el nuevo resultado.
6. El centro de la figura será siempre el punto medio del PictureBox.
7. Si el usuario desea eliminar el dibujo actual, puede utilizar el botón Limpiar.

El formulario dibuja directamente sobre el área gráfica y no conserva el dibujo al minimizar, salvo que se implemente un buffer adicional. Esto forma parte de la demostración y explicación técnica incluida por el estudiante.

#### **1.4. Parámetros utilizados**

1. cx, cy: coordenadas del centro calculadas automáticamente.
2. r: radio ingresado por el usuario.
3. pixelSize: escala visual del punto, definida por el usuario.
4. color: color elegido para el trazo.
5. Graphics g: superficie de dibujo.

En el método paramétrico se incluye un parámetro interno denominado paso, calculado como  $1/r$ , que evita la separación entre puntos.

#### **1.5. Análisis comparativo de las variantes**

1. Eficiencia

- Bresenham y Punto Medio son los más rápidos, basados en enteros.
- Paramétrico es más lento debido a coseno y seno.

## 2. Precisión

- Bresenham y Punto Medio producen contornos uniformes.
- Paramétrico depende del paso angular, pudiendo generar huecos si el paso es demasiado grande.

## 3. Suavidad

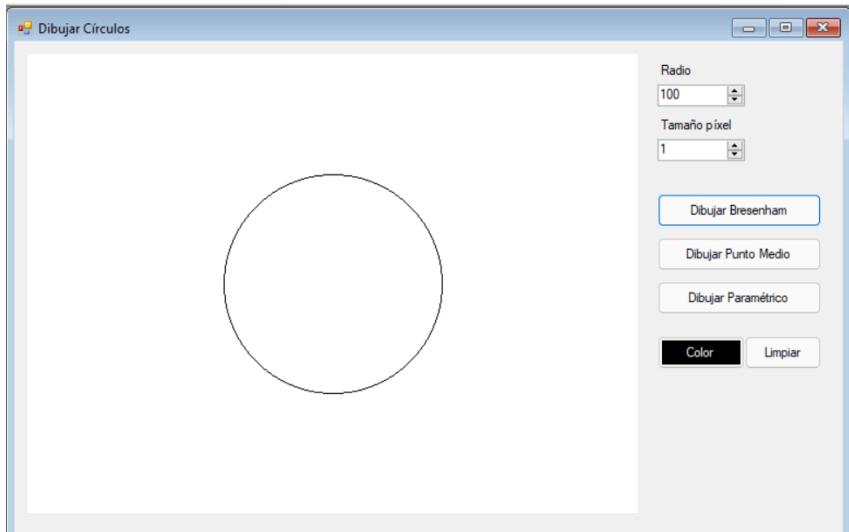
- Paramétrico puede mostrar un borde aparentemente más continuo dependiendo del tamaño del paso.
- Bresenham y Punto Medio se presentan más "pixeleadados" pero muy precisos.

## 4. Uso académico

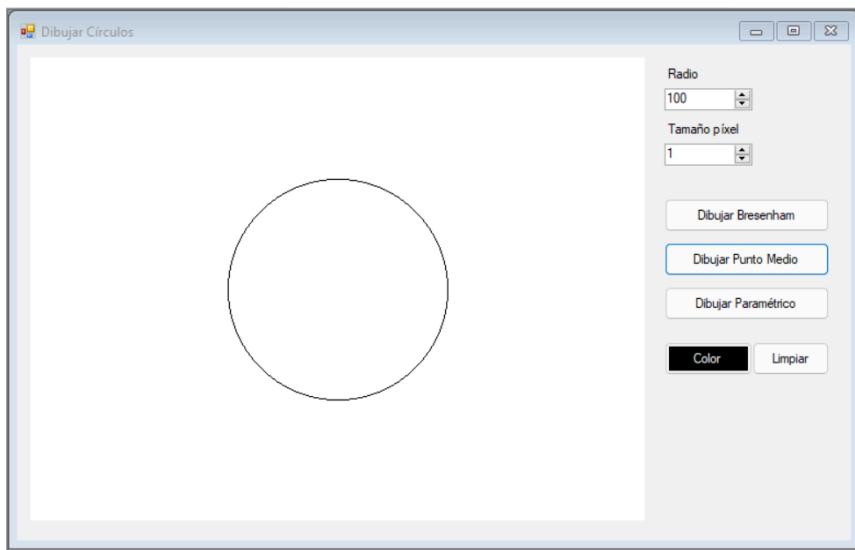
- Bresenham se utiliza para demostrar optimización.
- Punto Medio se utiliza para fundamentación geométrica.
- Paramétrico permite demostrar conceptos trigonométricos y control artístico.

## 1.6. Capturas de pantalla

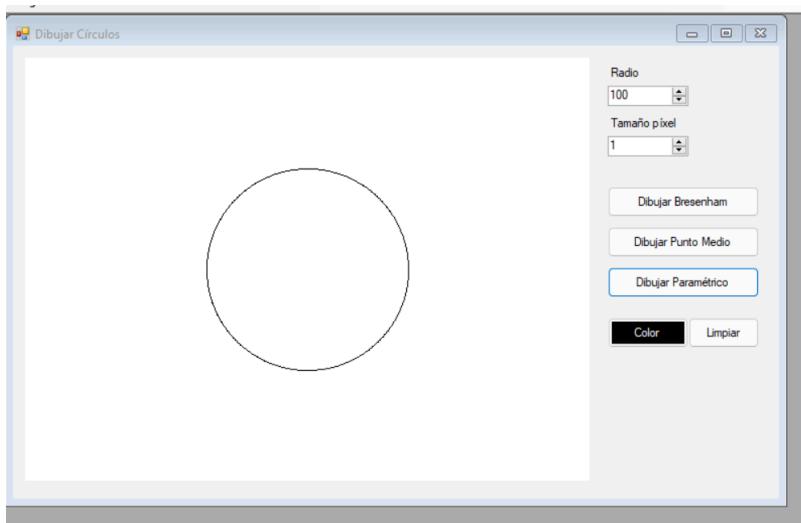
### 1. Ejecución del algoritmo Bresenham:



2. Ejecución del algoritmo Punto Medio:



3. Ejecución del algoritmo Paramétrico:



## **2.1. Algoritmo: Dibujo de Líneas**

### **2.1.1. Descripción técnica general**

Para el trazado de líneas rectas entre dos puntos arbitrarios se implementaron tres algoritmos clásicos: DDA (Digital Differential Analyzer), Bresenham y MidPoint. Cada uno utiliza una estrategia diferente para el cálculo de los pixeles que aproximan la línea, permitiendo analizar rendimiento, suavidad del trazo, complejidad del cálculo y comportamiento ante pendientes positivas, negativas o superiores a 1.

Los tres métodos trabajan sobre una superficie Graphics y emplean un mecanismo común: colocar un píxel mediante la rutina DrawRectangle(pen, x, y, 1, 1).

Las coordenadas ingresadas por el usuario en el formulario (FormLineas) se convierten a coordenadas en píxeles mediante un sistema cartesiano centrado, permitiendo ingresar valores positivos y negativos sin salir del área visible.

## **2.2. Variantes implementadas**

### **2.2.1. Variante 1: DDA (Digital Differential Analyzer)**

#### **a) Descripción técnica**

El algoritmo DDA calcula la línea mediante incrementos fraccionarios. Partiendo del punto inicial ( $x_0, y_0$ ), determina el número de pasos en función de la diferencia en los ejes  $dx$  y  $dy$ . Los incrementos por paso se calculan como:

$$xInc = dx / steps$$

$$yInc = dy / steps$$

Cada iteración avanza en estos incrementos acumulativos y traza un pixel redondeando las coordenadas generadas.

El método produce líneas suaves, especialmente cuando se usa una escala adecuada, porque emplea aritmética en coma flotante.

#### **b) Justificación**

Es uno de los algoritmos más utilizados al iniciar el estudio de rasterización por su fácil implementación y su claridad conceptual. Permite al estudiante entender cómo se genera una línea mediante pequeños incrementos desde un punto inicial a uno final.

#### **c) Caso de uso demostrado**

El estudiante valida el funcionamiento de DDA con líneas horizontales, verticales, diagonales y con pendientes mayores que 1. También verifica cómo el redondeo afecta la continuidad del trazo.

## 2.2.2. Variante 2: Bresenham

### a) Descripción técnica

El algoritmo de Bresenham utiliza únicamente operaciones enteras (sumas y restas). El método calcula la diferencia absoluta en los ejes ( $dx$  y  $dy$ ) y determina la dirección de avance mediante  $sx$  y  $sy$ . El valor  $err$  se emplea como error acumulado para decidir cuándo desplazar la coordenada vertical o horizontal.

La lógica principal se basa en:

$e2 = 2 * err$   
Si  $e2 > -dy \rightarrow$  ajustar horizontal  
Si  $e2 < dx \rightarrow$  ajustar vertical

El algoritmo continúa hasta alcanzar el punto final.

### b) Justificación

Es más eficiente que DDA porque evita operaciones en coma flotante. Su rendimiento y precisión son ideales para sistemas con recursos limitados o para aplicaciones donde se requiere velocidad. Es el algoritmo estándar en rasterización de líneas.

### c) Caso de uso demostrado

El estudiante muestra el trazado de líneas complejas, incluyendo pendientes negativas y valores grandes de  $dx$  y  $dy$ . Se evalúa cómo el algoritmo mantiene la continuidad del trazo sin importar la dirección de la línea.

## 2.2.3. Variante 3: MidPoint (Punto Medio)

### a) Descripción técnica

El algoritmo MidPoint evalúa un punto medio geométrico entre dos posibles posiciones del siguiente pixel.

Primero normaliza los puntos para garantizar que  $x_0 \leq x_1$ . Luego calcula  $dx$ ,  $dy$  y determina la dirección vertical mediante  $sx$ .

La decisión se basa en la variable  $d$ :

$$d = 2dy - dx$$

Si  $d \leq 0$ , la siguiente posición avanza hacia el Este (incremento en  $x$ ).

Si  $d > 0$ , avanza hacia el Noreste (incremento en  $x$  y  $y$ ).

En cada iteración se actualizan  $d$ ,  $incE$  y  $incNE$  según la ecuación implícita de la línea.

### b) Justificación

Es matemáticamente más sencillo que Bresenham, aunque su desempeño y apariencia visual son equivalentes. Se utiliza con frecuencia en educación debido a que ilustra claramente el funcionamiento del criterio geométrico.

### c) Caso de uso demostrado

El estudiante comprueba el funcionamiento con líneas de pendiente positiva, negativa y pendientes mayores que 1, validando que la normalización inicial evita trazados invertidos.

### 2.3. Instrucciones de uso del formulario (FormLineas)

1. Ingresar los valores numéricos de  $x_0$ ,  $y_0$ ,  $xf$  y  $yf$  en los cuadros de texto. Se aceptan números positivos, negativos y decimales.
2. El sistema valida automáticamente la entrada, permitiendo el uso de punto decimal o coma.
3. El origen del plano cartesiano se ubica en el centro del PictureBox. Las coordenadas negativas son visibles hacia la izquierda y hacia abajo según la conversión realizada.
4. El estudiante puede modificar la escala (scale) si requiere ampliar la visualización, aunque por defecto es 1.
5. Presionar uno de los botones:
  - DDA
  - Bresenham
  - MidPoint
6. El área se limpia y se dibuja una cuadrícula centrada para referencia.
7. El algoritmo seleccionado genera la línea utilizando las coordenadas convertidas.
8. El resultado aparece en el PictureBox junto con el sistema de ejes.

### 2.4. Parámetros utilizados

1.  $x_0, y_0$ : coordenadas iniciales ingresadas por el usuario.
2.  $xf, yf$ : coordenadas finales ingresadas por el usuario.

3. scale: cantidad de píxeles que representa una unidad del usuario.
4. originX, originY: punto central del área de dibujo.
5. g: superficie gráfica utilizada para rasterizar los píxeles.
6. p: PEN utilizado para la representación de cada punto.

Los tres algoritmos trabajan con los mismos parámetros ya convertidos a coordenadas de píxel mediante el método ToPixelCoordinates.

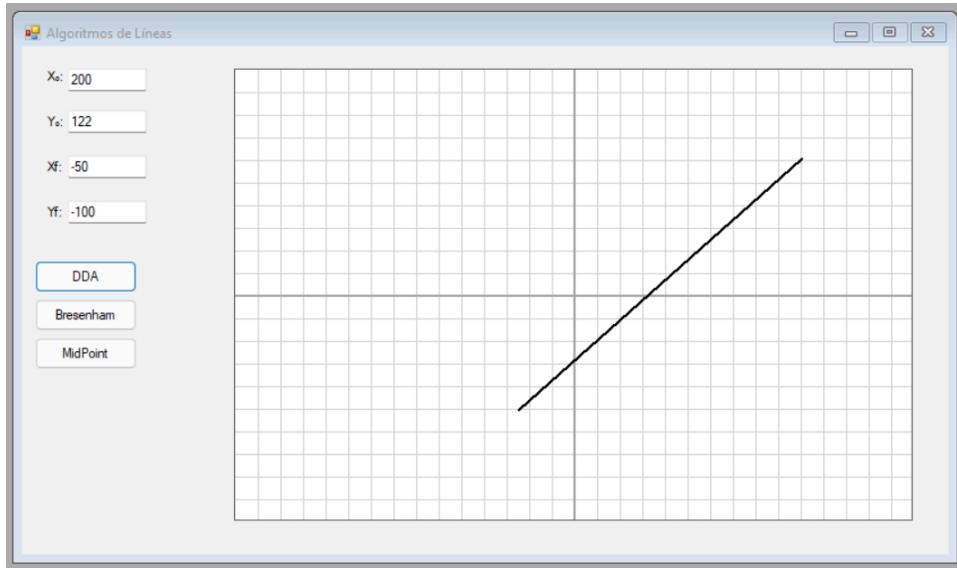
## 2.5. Análisis comparativo de las variantes

1. Eficiencia
  - Bresenham es el más eficiente porque usa únicamente enteros.
  - MidPoint tiene eficiencia similar pero requiere un paso adicional de normalización.
  - DDA es menos eficiente debido a la aritmética de punto flotante.
2. Suavidad del trazo
  - DDA puede generar suavidad visual mayor por el uso de redondeos fraccionarios.
  - Bresenham y MidPoint tienden a representar líneas rasterizadas más exactas pero con apariencia más marcada.
3. Robustez ante pendientes
  - Bresenham y DDA manejan sin dificultad pendientes negativas y pendientes mayores a 1.
  - MidPoint requiere normalización del orden para operar correctamente.
4. Uso académico
  - DDA muestra claramente el proceso incremental.
  - Bresenham demuestra optimización matemática e implementación industrial.

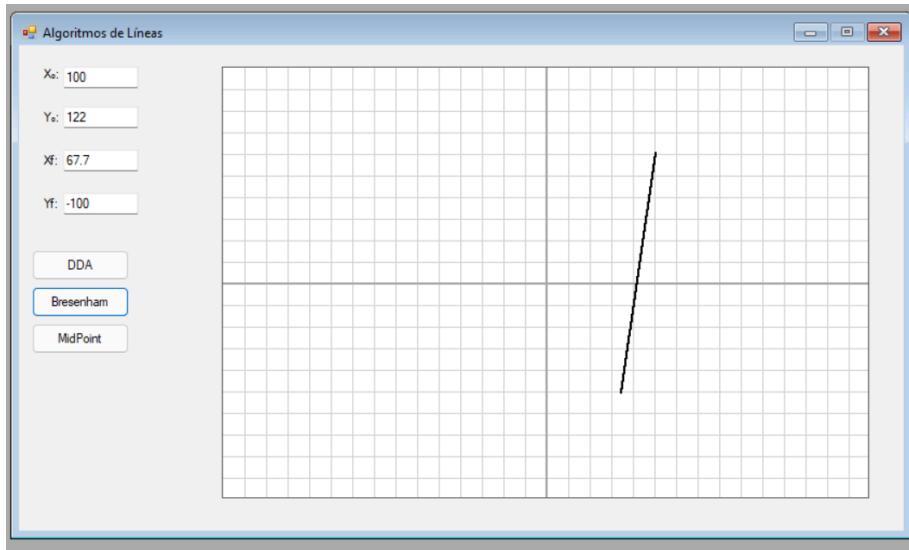
- MidPoint demuestra criterio geométrico y decisión por punto medio.

## 2.6. Capturas de pantalla

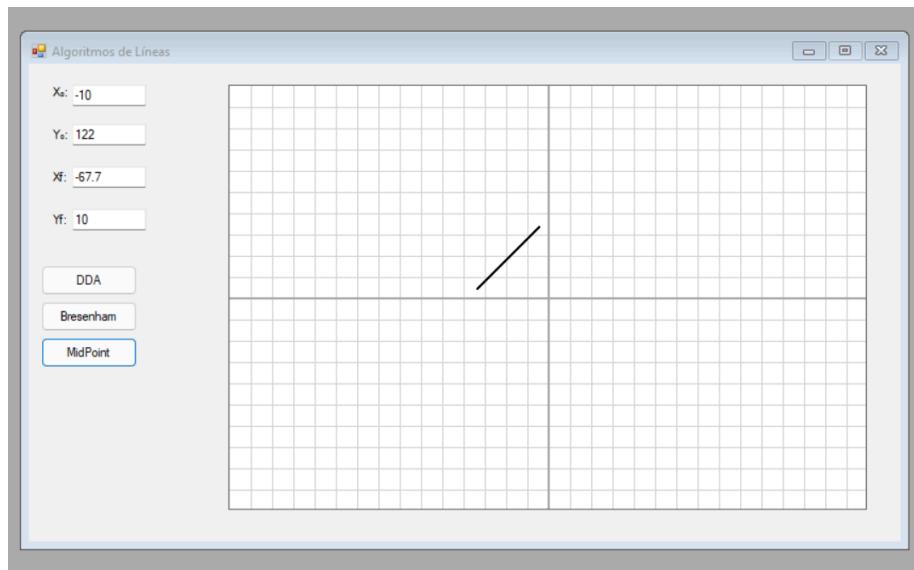
1. Ejecución del algoritmo DDA:



2. Ejecución de Bresenham:



### 3. Ejecución del algoritmo MidPoint:



#### 3.1. Algoritmo: Recorte de Polígonos (Cohen–Sutherland, Liang–Barsky y Sutherland–Hodgman)

##### 1.1. Descripción técnica general

Para el recorte de líneas y polígonos dentro de una ventana rectangular se implementaron tres algoritmos clásicos en gráficos por computadora: Cohen–Sutherland, Liang–Barsky y Sutherland–Hodgman. Cada uno utiliza un enfoque diferente para determinar qué segmentos permanecen visibles dentro de la región de recorte y cuáles deben ser descartados o ajustados.

El objetivo del recorte no es rellenar formas, sino **visualizar únicamente la porción del polígono o línea que queda dentro del rectángulo definido como ventana de recorte**, manteniendo el resto del dibujo con una intensidad reducida para referencia visual.

Las rutinas trabajan sobre una superficie Graphics, donde las líneas y polígonos se trazan mediante DrawLine o DrawPolygon. El usuario selecciona un polígono o curva y define la ventana rectangular de recorte. Las porciones internas se dibujan con color normal mientras que las externas reducen su intensidad.

#### 3.2. Variantes implementadas

##### 3.2.1. Variante 1: Cohen–Sutherland

###### a) Descripción técnica

Cohen–Sutherland utiliza una clasificación por regiones basada en códigos binarios llamados *outcodes*.

El plano alrededor del rectángulo de recorte se divide en nueve regiones. Cada punto obtiene un código de 4 bits que indica su posición relativa: izquierda, derecha, arriba, abajo.

A partir de estos códigos, el algoritmo decide:

- Si ambos puntos están dentro ( $\text{outcode} = 0$ ), la línea es completamente visible.
- Si la operación AND entre outcodes  $\neq 0$ , la línea está completamente fuera.
- En otros casos, se calcula una intersección con el borde correspondiente, se actualiza el punto y se repite el proceso hasta obtener un segmento recortado.

b) Justificación

Es uno de los algoritmos más usados históricamente por su rapidez para descartar segmentos enteros con operaciones lógicas. Su enfoque por regiones lo vuelve muy intuitivo para explicar recorte.

c) Caso de uso demostrado

El estudiante recorta líneas con diferentes posiciones, interceptando correctamente los bordes del rectángulo y mostrando solo la parte visible.

### 3.2.2. Variante 2: Liang–Barsky

a) Descripción técnica

Liang–Barsky usa una formulación paramétrica más eficiente.

La línea se describe como:

$$x = x_0 + t(dx)$$

$$y = y_0 + t(dy)$$

Luego, se calculan valores  $p$  y  $q$  que representan direcciones y límites. El recorte se obtiene restringiendo el parámetro  $t$  entre 0 y 1.

El algoritmo determina:

Si  $p = 0$  y  $q < 0 \rightarrow$  el segmento está completamente fuera.

Si  $p \neq 0 \rightarrow$  se actualizan  $t_0$  y  $t_1$  mediante  $q/p$ .

Al finalizar, el segmento recortado se encuentra evaluando los valores finales de  $t$ .

b) Justificación

Es más eficiente que Cohen–Sutherland porque evita iteraciones. Usa menos comparaciones y operaciones lógicas, logrando mejor rendimiento.

c) Caso de uso demostrado

El estudiante evalúa casos donde la línea entra y sale por varios lados del rectángulo. Liang–Barsky recorta con precisión incluso en líneas inclinadas o largas.

### **3.2.3. Variante 3: Sutherland–Hodgman**

#### a) Descripción técnica

Este algoritmo está diseñado para polígonos.

Recorta un polígono contra cada borde de la ventana, uno por uno, siguiendo este ciclo:

1. Recorte contra borde izquierdo
2. Recorte contra borde derecho
3. Recorte contra borde superior
4. Recorte contra borde inferior

En cada paso se genera una nueva lista de vértices.

La regla de inclusión se basa en evaluar cada pareja de vértices consecutivos, determinando si están dentro, fuera o cruzan el borde:

- Si ambos están dentro → se añade el segundo punto.
- Si el borde es cruzado → se añade el punto de intersección.
- Si ambos están fuera → no se añade nada.

#### b) Justificación

Es el estándar para recorte de polígonos convexos y funciona bien en muchos casos no convexos. Además, es simple y estructurado.

#### c) Caso de uso demostrado

El estudiante toma polígonos de diferentes formas (triángulos, hexágonos, polígonos irregulares) y verifica cómo el algoritmo conserva únicamente la sección dentro del rectángulo.

### **3.3. Instrucciones de uso del formulario (FormRecorte)**

Ingresar los vértices del polígono o los puntos de la línea en los cuadros de texto provistos.

El sistema permite:

- Valores positivos o negativos
- Valores decimales
- Ingreso punto a punto

Luego:

1. Definir la ventana de recorte mediante las coordenadas xmin, xmax, ymin, ymax.
2. Presionar uno de los botones:
  - Cohen–Sutherland (líneas)
  - Liang–Barsky (líneas)
  - Sutherland–Hodgman (polígonos)
3. Se dibuja el polígono original con un color tenue.
4. La parte visible dentro del rectángulo se dibuja con color brillante.
5. El rectángulo de recorte se muestra centrado en el área de dibujo para referencia.

El área se limpia automáticamente antes de cada trazo.

### **3.4. Parámetros utilizados**

coords: lista de puntos del polígono.

x0, y0, xf, yf: puntos de línea.

xmin, xmax, ymin, ymax: límites del rectángulo.

g: superficie gráfica utilizada.

penInterior: color de segmento visible.

penExterior: color tenue para segmentos externos.

scale: valor para convertir coordenadas reales a píxeles.

originX, originY: centro del PictureBox.

### **3.5. Análisis comparativo de las variantes**

Eficiencia

Liang–Barsky es el más eficiente para líneas.

Cohen–Sutherland realiza más pruebas pero descarta regiones rápidamente.

Sutherland–Hodgman se adapta mejor a polígonos.

Claridad conceptual

Cohen–Sutherland facilita entender las regiones.

Liang–Barsky es más matemático y paramétrico.

Sutherland–Hodgman explica bien el recorte secuencial por bordes.

Robustez

Los tres algoritmos manejan correctamente valores negativos, decimales y coordenadas fuera de rango.

Sutherland–Hodgman es el más estable con polígonos convexos.

Uso académico

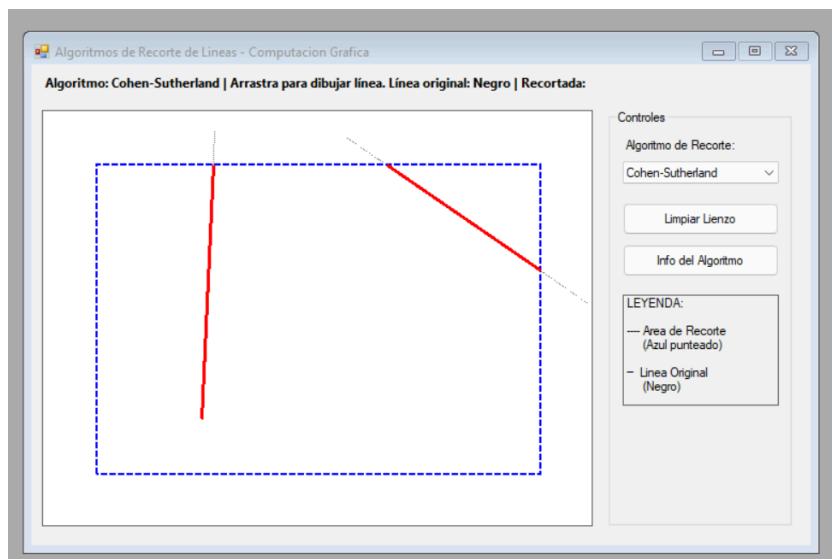
Cohen–Sutherland enseña límites regionales.

Liang–Barsky enseña parametrización.

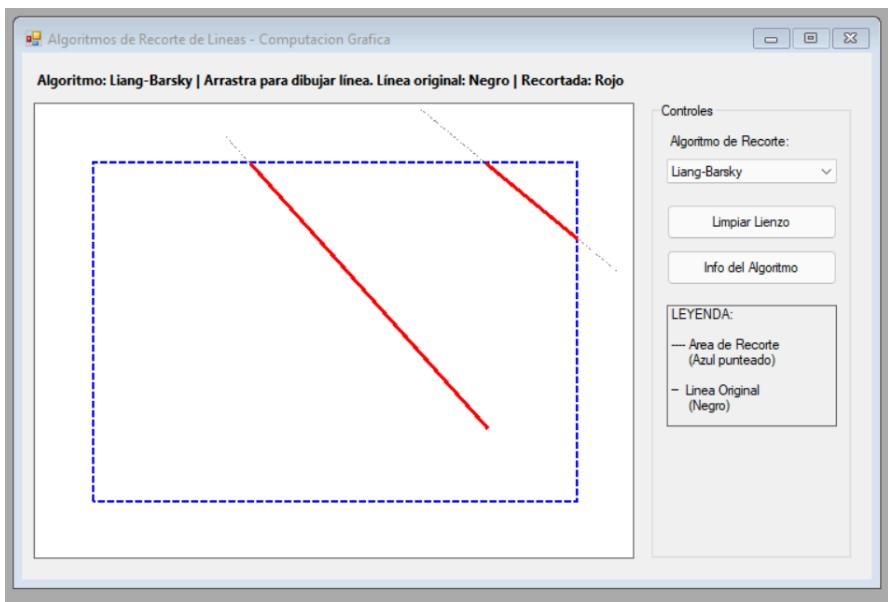
Sutherland–Hodgman enseña recorte iterativo de polígonos.

### 3.6. Capturas de pantalla.

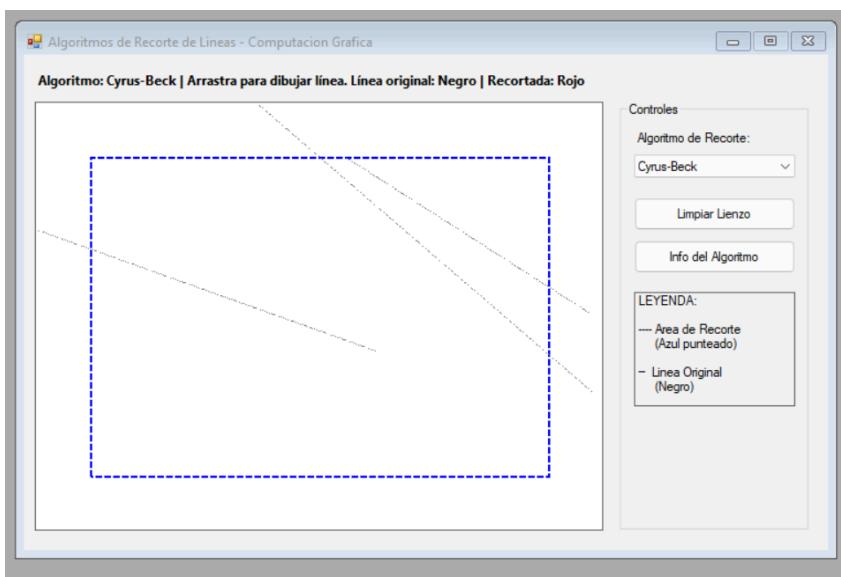
1. Ejecución Cohen–Sutherland:



2. Ejecución Liang–Barsky:



### 3. Ejecución Cyrus-Beck:



## 4.1 Algoritmo: Recorte de Polígonos (Clipping)

### 4.1.1. Descripción técnica general

El proceso de recorte de polígonos permite delimitar visualmente una región de interés dentro de una ventana rectangular. El objetivo es mostrar con total nitidez el contenido interior del rectángulo de recorte, mientras que el área exterior se representa con una disminución controlada de la intensidad del color, simulando una atenuación visual. El sistema implementado permite al usuario definir tanto la ventana rectangular de recorte como el polígono a visualizar.

El modelo matemático usado se basa en determinar qué puntos del polígono quedan dentro o fuera de la ventana, comparando cada coordenada con los límites izquierdo, derecho, superior e inferior. El sistema no rellena el polígono ni utiliza flood fill; únicamente aplica recorte geométrico y re-renderizado selectivo con ajuste de colores.

En el formulario (FormRecorte) el usuario dibuja un polígono punto a punto y posteriormente especifica un rectángulo de recorte. La lógica se ejecuta sobre un objeto Graphics mediante trazado individual de segmentos con DrawLine o trazado de puntos mediante DrawRectangle(pen, x, y, 1, 1) cuando se requiere resaltar los resultados.

#### **4.2. Variantes implementadas**

##### **4.2.1. Variante 1: Visualización normal (sin recorte)**

###### **a) Descripción técnica**

En esta modalidad el polígono se dibuja tal cual fue ingresado por el usuario, respetando el color seleccionado. No existe evaluación contra los límites del rectángulo de recorte. La superficie completa del PictureBox se muestra con el fondo original y el polígono se traza de manera continua.

###### **b) Justificación**

Es necesaria para que el usuario disponga de una referencia visual del estado inicial, evaluando la forma real del polígono antes del recorte.

###### **c) Caso de uso demostrado**

Se utiliza para validar la correcta captura de los vértices del polígono y para verificar que no exista distorsión en el trazado previo al proceso de clipping.

##### **4.2.2. Variante 2: Recorte por atenuación (clipping visual)**

###### **a) Descripción técnica**

En esta modalidad se dibuja el rectángulo de recorte y posteriormente se recorre cada segmento del polígono. Para cada punto que se encuentre fuera de la ventana, se reduce la intensidad del color aplicando un factor multiplicativo, generalmente entre 0.4 y 0.6, para simular una desaturación visual. Los puntos dentro de la ventana mantienen su color original.

La condición de pertenencia se verifica mediante:

x dentro si  $x \geq \min X$  y  $x \leq \max X$

y dentro si  $y \geq \min Y$  y  $y \leq \max Y$

Los puntos que no cumplen esta condición se atenúan.

###### **b) Justificación**

Permite visualizar de manera intuitiva qué porciones del polígono quedan fuera del área válida sin eliminar información. Es ideal para análisis gráfico educativo.

### c) Caso de uso demostrado

El estudiante compara cómo se atenúan los puntos externos y cómo se mantiene el color dentro de la ventana. Esto facilita el entendimiento del concepto sin perder la forma total del polígono.

#### 4.2.3. Variante 3: Recorte rígido (clip estricto)

##### a) Descripción técnica

En esta variante únicamente se dibujan los segmentos del polígono que se encuentran completamente dentro del rectángulo de recorte. Si un segmento se encuentra parcialmente dentro, se recorta matemáticamente calculando las nuevas intersecciones con los bordes de la ventana.

El algoritmo implementado es una simplificación del método de Cohen–Sutherland y Liang–Barsky. Para cada límite (izquierda, derecha, arriba, abajo) se calcula la intersección si el segmento cruza el borde. Los puntos resultantes reemplazan a los originales y solo los puntos dentro de la región se renderizan.

##### b) Justificación

Representa el clipping exacto, utilizado en renderizado profesional, motores gráficos y sistemas CAD. Permite comprender la geometría parcial de un polígono respecto a una ventana de visión.

##### c) Caso de uso demostrado

Se muestran polígonos que atraviesan la ventana, evidenciando cómo los segmentos externos desaparecen y únicamente se mantiene la porción interna exacta.

### 4.3. Instrucciones de uso del formulario (FormRecorte)

El usuario selecciona puntos dentro del PictureBox con el mouse para definir los vértices del polígono. Cada clic añade un punto a la lista interna. El polígono se cierra automáticamente si el usuario presiona el botón “Cerrar Polígono”.

Para definir el rectángulo de recorte, el usuario puede:

- ingresar coordenadas numéricamente (`minX`, `minY`, `maxX`, `maxY`), o
- dibujar el rectángulo arrastrando el mouse (según la variante del formulario que utilice).

Una vez definidos el polígono y el rectángulo, el usuario puede elegir:

- Visualizar sin recorte
- Aplicar recorte con atenuación
- Aplicar recorte rígido

El área se limpia, se dibuja el rectángulo de recorte y luego el polígono según la modalidad seleccionada.

#### **4.4. Parámetros utilizados**

vertices: lista de puntos ingresados por el usuario para el polígono.  
minX, minY, maxX, maxY: límites del rectángulo de recorte.  
g: superficie Graphics utilizada para el renderizado.  
colorOriginal: color seleccionado por el usuario.  
colorAtenuado: variante multiplicada del color original para zonas exteriores.  
intersecciones: puntos calculados durante el recorte rígido.  
p: PEN o SolidBrush para el trazado según la modalidad.

#### **4.5. Análisis comparativo de las variantes**

Eficiencia

La visualización normal es la más rápida.  
El recorte por atenuación agrega condiciones por punto, aumentando el costo ligeramente.  
El recorte rígido es el más complejo por los cálculos de intersección.

Claridad visual

La atenuación es la variante más ilustrativa para entender qué está dentro y qué está fuera.  
El recorte rígido es exacto pero elimina información externa.  
La visualización normal sirve únicamente como referencia previa.

Robustez ante polígonos irregulares

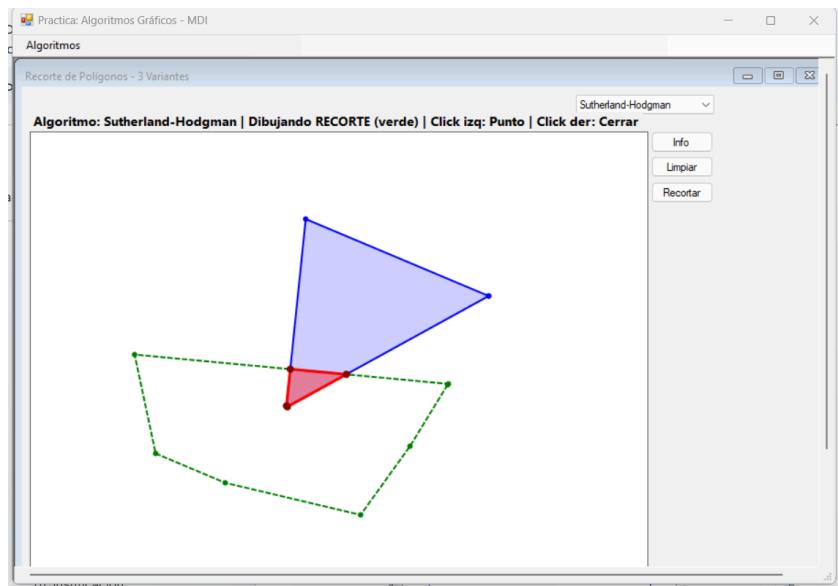
El sistema soporta polígonos abiertos y cerrados sin requerir orden estricto.  
El recorte rígido funciona correctamente si los segmentos intersectan la ventana al menos una vez.

Uso académico

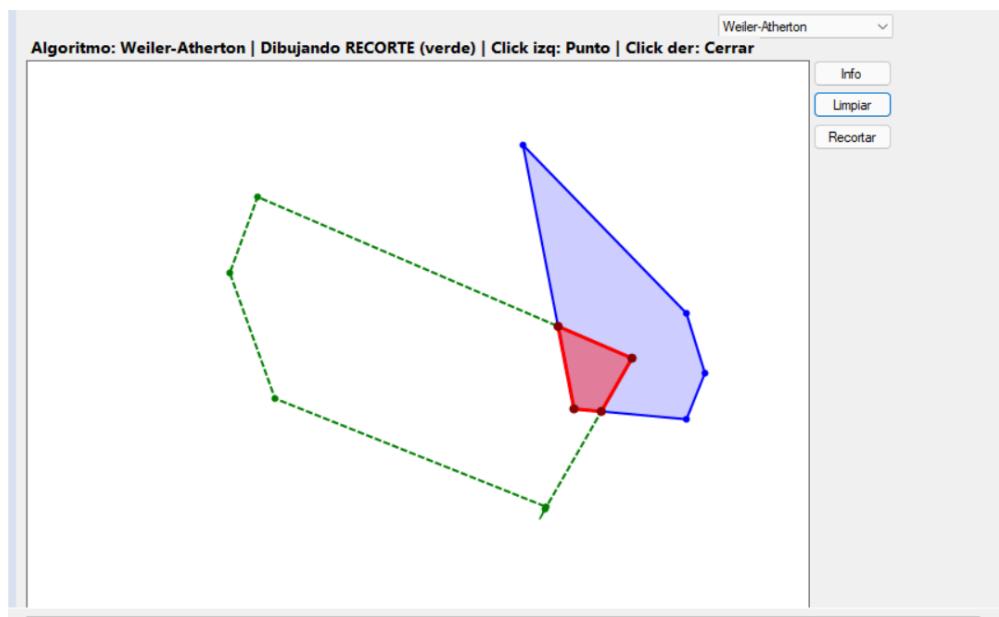
La atenuación permite comprender intuitivamente el clipping.  
El recorte rígido permite estudiar algoritmos clásicos de ventanas de visualización.  
Las tres variantes juntas ofrecen una visión completa del proceso.

#### **4.6. Capturas de pantalla:**

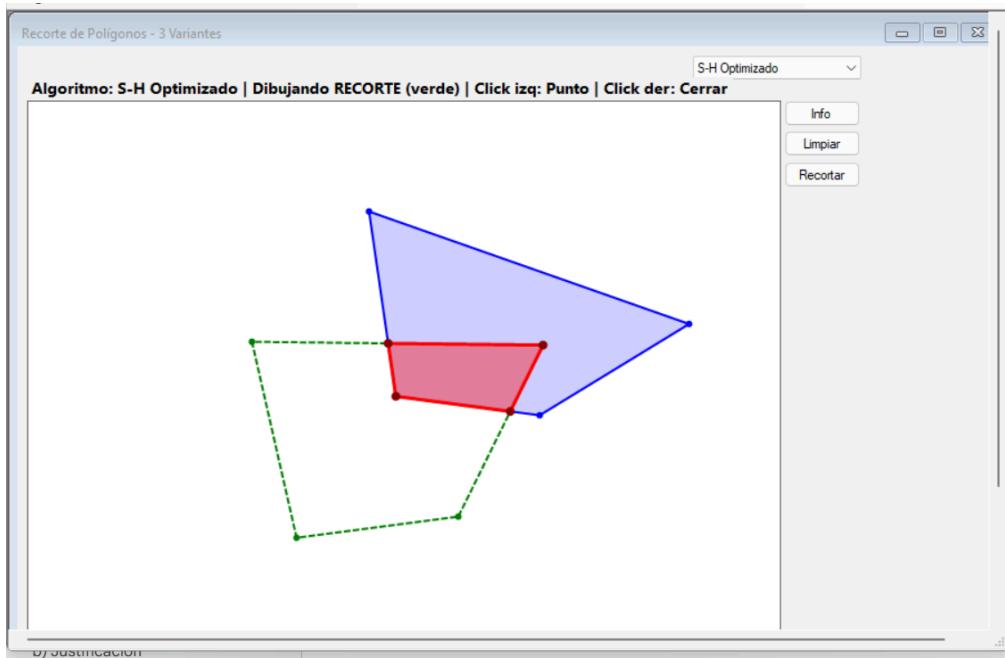
1. Ejecución Sutherland-Hodgman:



2. Ejecución Weiler-Atherton:



3. Ejecución S-H Optimizado:



## 5.2. Algoritmo: Relleno de Regiones

### 5.2.1. Descripción técnica general

Para el relleno de figuras bidimensionales rasterizadas se implementaron tres variantes del algoritmo Flood Fill: BFS (por amplitud), DFS (por profundidad iterativa) y Scanline Fill (por franjas horizontales).

Cada técnica utiliza estrategias diferentes para expandir el color hacia los píxeles adyacentes, permitiendo estudiar eficiencia, estabilidad, tolerancia a áreas grandes y resistencia a overflow.

Las tres variantes trabajan directamente con un objeto Bitmap, utilizando operaciones GetPixel() y SetPixel() para leer y escribir colores en memoria de imagen.

El usuario indica el punto inicial (semilla) mediante un clic o introduciendo coordenadas manualmente. El algoritmo sustituye el color original detectado en la posición semilla por un nuevo color definido por el usuario.

El formulario (FormRelleno) permite dibujar figuras básicas (rectángulo, círculo) y polígonos a mano alzada, que luego pueden ser llenados utilizando cualquiera de las tres variantes. Esto facilita comparar el comportamiento del relleno en formas cerradas y abiertas.

## 5.3. Variantes implementadas

### 5.3.1. Variante 1: Flood Fill BFS (Cola)

#### a) Descripción técnica

La variante BFS (Breadth-First Search) utiliza una cola para recorrer el área de relleno en todas las direcciones horizontales y verticales.

El procedimiento:

1. Se obtiene el color original desde el punto semilla.
2. Se coloca ese punto dentro de una cola.
3. Iterativamente se extraen puntos de la cola, se colorean y se agregan sus vecinos directos (arriba, abajo, izquierda, derecha).
4. El proceso continúa hasta que la cola queda vacía.

El recorrido por amplitud garantiza que el relleno avanza uniformemente hacia afuera desde el punto inicial, lo que resulta en una expansión ordenada.

### **b) Justificación**

Es un método muy estable y conceptualmente simple.

Funciona bien para figuras cerradas, incluso complejas, aunque no es el más rápido debido al uso intensivo de memoria en la cola.

### **c) Caso de uso demostrado**

El estudiante rellena rectángulos, círculos y polígonos irregulares para observar cómo el algoritmo expande el color capa por capa.

Es especialmente útil para entender el comportamiento de la expansión por niveles.

#### **5.3.2. Variante 2: Flood Fill DFS Iterativo (Pila)**

##### **a) Descripción técnica**

La variante DFS (Depth-First Search) se implementa con una pila para evitar los errores típicos del método recursivo (StackOverflow).

Su funcionamiento:

1. Lee el color original en la semilla.
2. Inserta ese punto en la pila.
3. Mientras la pila contiene elementos, extrae un punto y colorea.
4. Inserta los vecinos directos.

El recorrido en profundidad provoca que el algoritmo avance en líneas largas antes de regresar a nodos pendientes.

### b) Justificación

Es más rápido que BFS en áreas grandes.

Además, evita por completo las limitaciones de recursividad de la versión clásica.

### c) Caso de uso demostrado

Se utiliza para llenar círculos grandes y polígonos complejos donde BFS puede tardar más tiempo.

Permite observar la expansión tipo "túnel" característica de DFS.

## 5.3.3. Variante 3: Flood Fill Scanline (Franja horizontal)

### a) Descripción técnica

La variante Scanline es la más eficiente.

La idea principal:

1. Desde el punto semilla, se colorea toda la línea horizontal hacia izquierda y derecha hasta límites con otro color.
2. Para cada píxel de esa franja, se revisan las líneas superior e inferior.
3. Si existe un segmento con color original, se inserta su punto inicial en la pila.
4. El proceso continúa expandiéndose línea por línea.

Esta técnica minimiza operaciones GetPixel() y SetPixel(), colorea franjas completas y reduce drásticamente la exploración redundante.

### b) Justificación

Es el método ideal para áreas muy grandes, pues es significativamente más veloz que BFS y DFS.

Se utiliza en software profesional debido a su alto rendimiento.

### c) Caso de uso demostrado

El estudiante rellena un círculo grande y observa la eficiencia del método, comparándolo con los tiempos de BFS y DFS.

Scanline completa la tarea con menor consumo de memoria y mayor velocidad.

#### **5.4. Instrucciones de uso del formulario (FormRelleno)**

1. Dibujar una figura:
  - Rectángulo
  - Círculo
  - Polígono (activando la opción correspondiente y haciendo clics para agregar vértices).
2. Si se trata de un polígono, cerrar la figura usando el botón “Cerrar Polígono”.
3. Seleccionar el color de relleno mediante el botón de selección de color.
4. Seleccionar uno de los algoritmos en la lista desplegable:
  - Flood Fill BFS
  - Flood Fill DFS
  - Flood Fill Scanline
5. Escoger un punto de inicio:
  - Haciendo clic directamente sobre el área de dibujo
  - O escribiendo las coordenadas X, Y manualmente
6. Presionar “Rellenar” para aplicar el algoritmo.
7. En caso necesario, utilizar “Limpiar” para reiniciar el lienzo.

El formulario actualiza automáticamente la imagen al finalizar cada operación.

#### **5.5. Parámetros utilizados**

bmp: imagen contenedora donde se realiza el relleno.

x, y: coordenadas de la semilla inicial.

nuevoColor: color seleccionado por el usuario.

original: color detectado en la semilla, que será reemplazado.

cola/pila: estructuras para BFS y DFS.

left/right: límites horizontales usados en Scanline.

Dentro(): función auxiliar para evitar accesos fuera del bitmap.

## 5.6. Análisis comparativo de las variantes

Rendimiento

Scanline es claramente el más rápido.

DFS es intermedio.

BFS es el más lento.

Uso de memoria

BFS consume más memoria debido a la cola.

DFS consume menos.

Scanline usa muy poca memoria.

Estabilidad

BFS es muy estable.

DFS es rápido pero puede re-explorar zonas si la figura tiene cavidades.

Scanline requiere formas relativamente bien definidas.

Relleno visual

Los tres métodos producen resultados idénticos en términos de color final, variando únicamente en la velocidad.

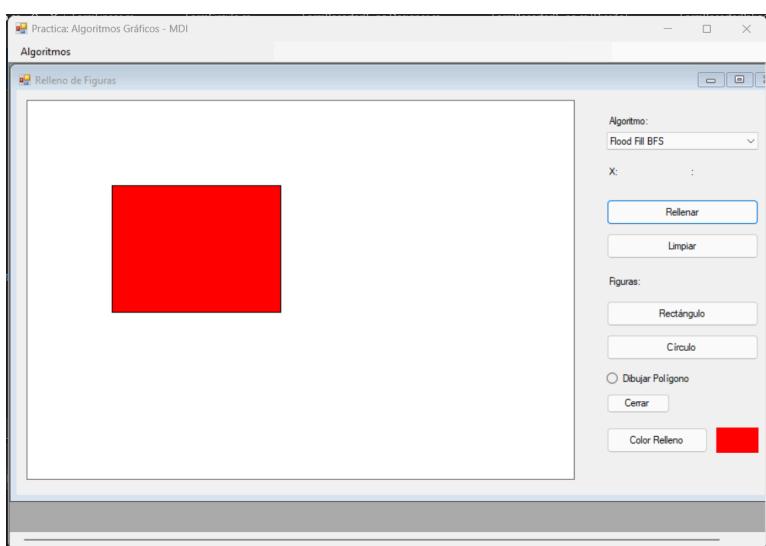
Uso académico

BFS y DFS permiten entender recorridos en grafos.

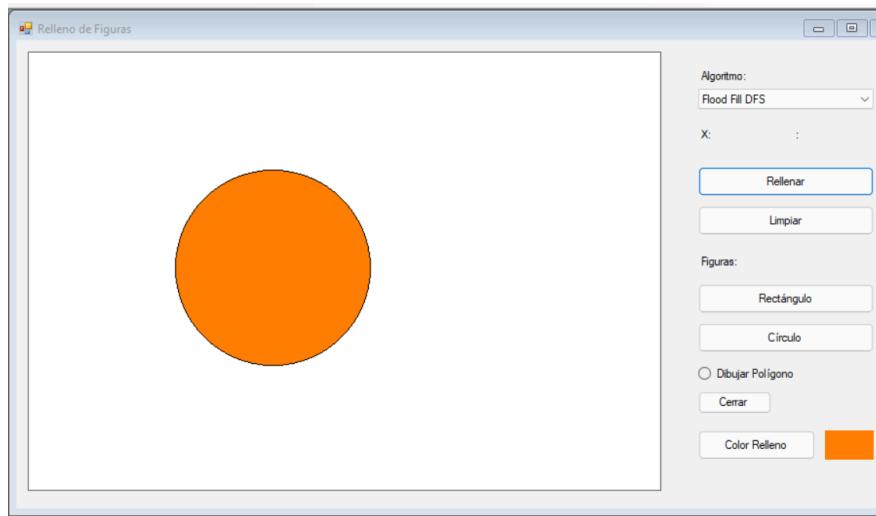
Scanline introduce el concepto de procesamiento por franjas, usado en gráficos avanzados.

## 5.7. Capturas de Pantalla

### 1. Ejecución Flood Fill BFS:



### 2. Ejecución Flood Fill DFS:



### 3. Ejecución Flood Fill Scanline:

