ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

PROYECTO FINAL #2

IMPRESORA 3D

INTEGRANTES: CORONADO MATEO

QUIZHPE EDISON

SANI NICOLE

ZÚÑIGA LIAM

INGENIERO: ZEA JONATHAN

MATERIA: MÉTODOS NUMÉRICOS

CURSO: GR1CC

FECHA DE ENTREGA: 11/02/2025

PERIODO

2024-B

I. INDICE

II. OBJETIVO GENERAL

- Desarrollar una impresora en 3D que nos permita simular el funcionamiento en un plano 2D a partir de la implementación de un archivo SVG y parámetros de entrada.

III. OBJETIVOS ESPECÍFICOS

- Establecer una interfaz de usuario que nos permita configurar los parámetros de impresión necesarios cómo la resolución de impresión (distancia entre los puntos de deposición) y la velocidad.
- Demostrar el movimiento de la impresora en las direcciones X y Y, a través de ecuaciones paramétricas.
- Aplicar métodos de optimización matemática que nos ayuden a encontrar la configuración más eficiente en función de los parámetros ingresados.
- Simular la trayectoria que recorre la impresora con los parámetros que se hayan ingresado, representando así su recorrido en un plano bidimensional.
- Desarrollar matemáticamente cómo se encuentra la trayectoria que debe recorrer la impresora con los parámetros establecidos.

IV. INTRODUCCIÓN

La impresora 3D fue creada en 1981 por el japonés Hideo Kodama, esta funciona principalmente cómo una impresora normal de tinta, pero en vez de depositar tinta, esta lo que hace es depositar materiales cómo plásticos, resinas, metales o termoplásticos en una serie de capas sucesivas con el cabezal que trabaja de abajo hacia arriba para crear el objeto que se solicite mediante un plano o formato digital realizado por un ordenador, estas impresoras son bastante usadas para la fabricación de objetos tridimensionales, nos pueden ayudar a realizar prototipos, productos finales y hasta piezas industriales, que es para lo que actualmente más se usa.

Esta impresora contiene un cabezal que es el que se encarga de construir el objeto, pero para esta construcción se programa el recorrido que este debe seguir gracias al conjunto de instrucciones que el usuario generaría de acuerdo con el diseño que se quiera implementar, en este proyecto lo que buscamos es simular el comportamiento de esta impresora pero en un entorno 2D basándonos en un archivo proporcionado de SVG y parámetros específicos que se pueden configurar para observar cómo se comporta nuestra impresora realizando diferentes cambios que influirán en el resultado final cómo la calidad de la impresión y el tiempo utilizado.

A través de la simulación debemos encontrar y calcular la trayectoria que debe seguir el cabezal teniendo en cuenta las características del archivo proporcionado el cuál se encarga de definir la geometría del objeto que queramos realizar y esta trayectoria se debe calcular matemáticamente usando métodos numéricos que nos permitan conseguir el recorrido más preciso y el más eficaz en función de los parámetros ingresados todo esto representado en un plano bidimensional lo que nos permitirá tener una mejor evaluación.

En este proyecto haremos uso de los SVG que se nos proporcionaron para trabajar, los SVG son un formato de imagen vectorial basado en XML, a diferencia de una imagen normal estos guardan la información de cada píxel, y en este caso nos ayuda a definir la ruta que tenemos definida de manera matemática, obtener la geometría del objeto en 2D, Transformar y extraer datos para la simulación y Definir la trayectoria de la impresión, en sí actuar como la fuente de datos geométricos que nos define al objeto.

La creación de mallas (Grillas 2D) o la generación de mallas discretizan en una superficie geométrica o un volumen en múltiples elementos. Durante la creación de mallas, las superficies 2D se representan mediante una sucesión de puntos ordenados a

través de un plano, para este caso se tomó la fórmula de progresión aritmética para la generación de la malla estructurada por la cual va a atravesar la impresión.

Punto en un polígono, en geometría computacional, el problema de punto en polígono (PIP) pregunta si un punto dado en el plano se encuentra dentro, fuera o en el límite de un polígono. La solución consiste en dibujar semi-líneas (o rayos) que pasen por P y contar el número de veces que el rayo intercepta con el polígono. Si el número de veces es impar, entonces el punto está dentro del polígono. Si el número es par, entonces el punto está fuera.

V. METODOLOGÍA

- **Descripción de la solución**

La primera etapa consiste en la creación de una malla bidimensional (grilla 2D) que cubre el área de interés. Para ello, se define un espacio de trabajo delimitado por coordenadas mínimas y máximas en los ejes X e Y. La malla se construye utilizando un paso (Δx, Δy) que determina la resolución de la discretización. Cada punto de la malla representa una posible posición en el espacio de trabajo.

A continuación, se realiza la verificación de puntos dentro del polígono, proceso en el cual se determina qué puntos de la malla se encuentran dentro del área definida. Para lograr esto, se emplea el Teorema del Número de Cruces, que evalúa si un punto está dentro del polígono contando el número de intersecciones entre una línea proyectada desde dicho punto y los bordes del polígono. Si el número de cruces es impar, el punto se encuentra dentro; si es par, está fuera.

Para optimizar la exploración del área, se implementa un algoritmo de escaneo en zigzag, lo que permite recorrer los puntos dentro del polígono de manera eficiente. Este método consiste en moverse a lo largo de filas o columnas alternando la dirección en cada paso, minimizando movimientos innecesarios y optimizando el tiempo total del recorrido. Esta estrategia es ampliamente utilizada en impresoras 3D y procesos de manufactura digital, donde es crucial garantizar una cobertura uniforme con el menor número de movimientos sin utilidad.

En la parte de código, la solución desarrollada es una aplicación en Python con una interfaz gráfica basada en Tkinter, diseñada para simular y visualizar el proceso de impresión en una impresora 3D. La aplicación permite a los usuarios cargar archivos SVG que contienen trayectorias de impresión, interpretar sus datos y representar gráficamente el movimiento de la impresora mediante una animación interactiva en Matplotlib. Esta representación facilita la comprensión del recorrido de la boquilla de impresión y ayuda a evaluar la precisión y eficiencia del diseño antes de llevarlo a la fabricación real.

El sistema implementa un procesamiento automatizado de archivos SVG utilizando svgpathtools, extrayendo las trayectorias vectoriales y convirtiéndolas en coordenadas que se pueden graficar. Además, el usuario puede ajustar parámetros clave, como la resolución de impresión y la velocidad de animación, para personalizar la simulación según sus necesidades. La aplicación valida estos parámetros para evitar valores incorrectos que puedan afectar la representación gráfica.

Para mejorar la experiencia del usuario, la interfaz incorpora elementos intuitivos, como botones para seleccionar archivos, modificar colores de visualización y controlar la animación. También se incluye una pantalla de carga con una barra de progreso simulada, proporcionando retroalimentación visual mientras el programa se inicia. Estas características hacen que la herramienta sea accesible tanto para principiantes como para usuarios avanzados en impresión 3D.

La animación de la trayectoria se logra mediante Matplotlib.animation, lo que permite visualizar el movimiento de la impresora en tiempo real. Se han implementado configuraciones de escala y límites de ejes para asegurar que la representación sea clara y fiel a la realidad. Además, el sistema permite resaltar el contorno y el relleno del diseño con colores personalizables, facilitando la identificación de cada segmento de la impresión.

- **Desarrollo matemático**

Creación de Mallas (Grillas 2D)

Para el desarrollo se determinaron varios conceptos como la creación de mallas, los cuales permite resolver el problema planteado, donde se crea una cuadrícula de puntos distribuidos de manera uniforme con un espaciado fijo y se dan en base a los conceptos mencionados anteriormente:

$$x_i = x_{min} + i * d_x$$

$$y_j = y_{min} + j * d_y$$

Donde:

$$x_{min} = Coordenada\ mínima\ en\ el\ eje\ X$$

$$y_{min} = Coordenada\ mínima\ en\ el\ eje\ y$$

$$d_x = rson\ las\ distancias\ entre\ puntos\ en\ el\ eje\ x\ (resolución\ de\ impresión)$$

$$d_y = son\ las\ distancias\ entre\ puntos\ en\ el\ eje\ y\ (resolución\ de\ impresión)$$

$$i = Índice\ en\ la\ dirección\ X$$

$$j = Índice\ en\ la\ dirección\ y$$

Verificación de Puntos Dentro de un Polígono

El siguiente paso es decidir qué puntos de la malla están **dentro** de la figura para ser parte del relleno por lo cual se usa el concepto de punto en un polígono que dice:

$$Dado\ un\ polígono\ \boldsymbol{P}\ definido\ por\ vértices\ (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

$$Un\ punto\ \boldsymbol{P}\ se\ considera\ dentro\ si:$$

$$C(x, y) = \sum_{i=1}^{n} I((x\_i, y\_i), (x_{i+1}, y_{i+1}), (x, y))$$

Donde:

$$I(A, B, P) = \begin{cases} 1, si\ el\ segmento\ AB\ es\ cruzado\ por\ el\ rayo\ horizontal\ desde\ P \\ 0, \quad en\ caso\ contrario \end{cases}$$

Optimización de la Trayectoria

Después se debe implementar el patrón de la gráfica, para lo cual se retoma el concepto de la creación de mallas, pero con cambios para no generar saltos innecesarios.

Sube en una columna:

$$(x, y) \rightarrow (x, y + dy)$$

Baja una columna:

$$(x + dx, y) \rightarrow (x + dx, y - dy)$$

Simulación de impresión

Si se considera una función $f(t)$ que describe la evolución de una variable en el tiempo, entonces el método discretiza la evolución en intervalos Δt:

$$X_n = F(X_{n-1}, t_n)$$

Donde:

$$X_n \; representa \; el \; estado \; del \; sistema \; en \; el \; paso \; n.$$

$$F \; es \; una \; función \; que \; define \; la \; dinámica \; del \; sistema.$$

$$t_n = t_0 + n \; * \; \Delta t \; es \; el \; tiempo \; en \; el \; paso \; n.$$

$$\Delta t \; es \; el \; intervalo \; entre \; cuadros \; en \; la \; animación.$$

- ## Documentación

Este código implementa una aplicación gráfica en Python utilizando Tkinter y Matplotlib. Su objetivo es cargar archivos SVG, extraer sus contornos y simular el proceso de impresión 3D desde la parte superior, haciendo que se observe en dos dimensiones.

Importaciones:

```
import tkinter as tk
import numpy as np
import svgpathtools
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from tkinter import messagebox, colorchooser
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.path import Path
import time
```

- **tkinter:** Biblioteca estándar de Python para interfaces gráficas.
- **numpy:** Biblioteca para cálculos matemáticos y manipulación de arreglos.
- **svgpathtools:** Permite trabajar con archivos SVG y extraer sus trayectorias.
- **matplotlib.pyplot:** Para la generación de gráficos.
- **matplotlib.animation:** Para animaciones en Matplotlib.
- **time:** Biblioteca estándar para manejar pausas y tiempos de espera.
- **tkinter.messagebox y tkinter.colorchooser**: Para mostrar mensajes emergentes y seleccionar colores en la interfaz.
- **matplotlib.backends.backend_tkagg:** Para integrar Matplotlib en Tkinter.
- **matplotlib.path.Path:** Para verificar si un punto está dentro de una forma.

Función show_loading_screen()

```
def show_loading_screen():

    """Muestra una pantalla de carga con una barra de progreso."""

    loading_root = tk.Tk()
    loading_root.overrideredirect(True)
    loading_root.configure(bg='black')
    loading_root.geometry("300x150+500+300")

    loading_label = tk.Label(loading_root, text="Cargando... 0%", font=("Arial",
14), fg='white', bg='black')
    loading_label.pack(expand=True)

    def update_progress():

        """Actualiza la barra de progreso."""
```

```
        for i in range(1, 101):
            loading_label.config(text=f"Cargando... {i}%", fg='red')
            loading_root.update()
            time.sleep(0.02)  # 2 segundos en total (100 * 0.02)
        loading_root.destroy()

    loading_root.after(100, update_progress)
    loading_root.mainloop()
```

Muestra una pantalla de carga antes de iniciar la aplicación.

- Crea una ventana de Tkinter sin bordes (overrideredirect(True)).
- Muestra un texto de progreso (Cargando... 0%).
- Se actualiza en un bucle con update(), aumentando el porcentaje hasta completar el 100%.
- La pantalla de carga se cierra con destroy().

Función load_svg()

```
def load_svg():

    """Carga un archivo SVG y procesa sus trayectorias, para poder extraer el
contorno."""

    file_name = svg_var.get()
    if not file_name:
        return
    file_path = f"./models/{file_name}"
    paths, _ = svgpathtools.svg2paths(file_path)
    process_svg(paths, file_name)
```

Carga un archivo SVG seleccionado por el usuario y extrae su contorno.

- Obtiene el nombre del archivo desde un StringVar().
- Construye la ruta del archivo en ./models/.
- Usa svgpathtools.svg2paths() para extraer las trayectorias del SVG.
- Llama a process_svg() para procesar la forma.

Función process_svg(paths, file_name)

```
def process_svg(paths, file_name):

    """
    Extrae los puntos de contorno de un SVG y los envía para llenado.

    Args:
        paths (list): Lista de rutas extraídas del SVG.
        file_name (str): Nombre del archivo SVG procesado.
    """

    contour = []
    for path in paths:
        for segment in path:
            points = np.linspace(0, 1, 100)
            for point in points:
```

```
                pos = segment.point(point)
                contour.append((pos.real, pos.imag))
    fill_shape(contour, file_name)
```

Procesa las trayectorias del archivo SVG y extrae los puntos del contorno.

- Recorre cada trayectoria (path) y cada segmento (segment).
- Divide cada segmento en 100 puntos con np.linspace().
- Extrae las coordenadas (x, y) y las almacena en contour.
- Llama a fill_shape() para generar un patrón de relleno dentro de la forma.

Función validate_resolution()

```
def validate_resolution():

    """
    Valida la distancia entre puntos ingresada por el usuario.

    Returns:
        float: Valor de distancia si es válido, None si es inválido.
    """

    try:
        value = float(resolution_entry.get())
        if 0.1 <= value <= 0.5:
            return value
        else:
            messagebox.showerror("Error", "La distancia debe estar entre 0.1 y 0.5
mm.")
            return None
    except ValueError:
        messagebox.showerror("Error", "Ingrese un número válido.")
        return None
```

Valida la distancia entre puntos ingresada por el usuario.

- Intenta convertir el valor del Entry en un número flotante.
- Verifica que esté entre 0.1 y 0.5 mm.
- Muestra un error con messagebox.showerror() si el valor es inválido.

Función validate_speed()

```
def validate_speed():

    """
    Valida la velocidad de animación ingresada por el usuario.

    Returns:
        int: Velocidad si es válida, None si es inválida.
    """

    try:
        value = int(speed_entry.get())
        if value > 0:
            return value
        else:
```

```
            messagebox.showerror("Error", "La velocidad debe ser un número
positivo.")
            return None
    except ValueError:
        messagebox.showerror("Error", "Ingrese un número válido para la
velocidad.")
        return None
```

Valida la velocidad de animación ingresada.

- Intenta convertir el valor del Entry en un número entero.
- Verifica que sea mayor a 0.
- Si el valor es inválido, muestra un mensaje de error.

Función choose_color()

```
def choose_color():

    """Permite al usuario seleccionar un color para los puntos de relleno."""

    global point_color
    color = colorchooser.askcolor()[1]
    if color:
        point_color = color
```

Permite al usuario seleccionar un color para los puntos de relleno.

- Usa colorchooser.askcolor() para abrir el selector de colores.
- Si el usuario elige un color, lo almacena en la variable global point_color.

Función fill_shape(contour, file_name)

```
def fill_shape(contour, file_name):

    """
    Genera un patrón de relleno dentro de la forma delimitada por el contorno.

    Args:
        contour (list): Lista de coordenadas que forman el contorno.
        file_name (str): Nombre del archivo SVG procesado.
    """

    x, y = zip(*contour)
    point_spacing = validate_resolution()
    if point_spacing is None:
        return
    path = Path(contour, closed=True)
    filling_points = []

    if file_name == "casa.svg":
        x_range, y_range = (0.5, 7.5), (1, 7.5)
    else:
        x_range, y_range = (0, 10), (-1, 8)

    x_grid = np.arange(x_range[0], x_range[1], point_spacing)
    y_grid = np.arange(y_range[0], y_range[1], point_spacing)
    direction = 1
    for x_val in x_grid:
```

```
        y_sorted = y_grid if direction == 1 else y_grid[::-1]
        for y_val in y_sorted:
            if path.contains_point((x_val, y_val)):
                filling_points.append((x_val, y_val))
        direction *= -1

    animate_trajectory(contour, filling_points, x_range, y_range)
```

Genera un patrón de relleno dentro de la figura definida por contour.

- Define el rango de coordenadas:
    - Si el archivo es "casa.svg", (porque se descuadra) usa (0.5, 7.5) en x y (1, 7.5) en y.
    - Si es otro archivo, usa (0, 10) en x y (-1, 8) en y.
- Crea una malla de puntos (x_grid, y_grid) con np.arange().
- Verifica qué puntos están dentro de la figura usando path.contains_point().
- Alterna la dirección del recorrido en cada fila para mejorar la eficiencia.
- Envía los puntos generados a animate_trajectory() para su animación.

Función animate_trajectory(contour, filling_points, x_range, y_range)

```
def animate_trajectory(contour, filling_points, x_range, y_range):

    """
    Anima el relleno de la forma siguiendo el patrón de trayectoria.

    Args:
        contour (list): Lista de coordenadas del contorno.
        filling_points (list): Puntos de relleno generados.
        x_range (tuple): Rango del eje X.
        y_range (tuple): Rango del eje Y.
    """

    ax.clear()
    x_contour, y_contour = zip(*contour)
    x_fill, y_fill = zip(*filling_points) if filling_points else ([], [])

    ax.plot(x_contour, y_contour, color='blue')
    scatter = ax.scatter([], [], color=point_color, s=2)
    ax.set_xlim(x_range)
    ax.set_ylim(y_range)

    speed = validate_speed()
    if speed is None:
        return

    def update(frame):
        if frame < len(x_fill):
            scatter.set_offsets(np.c_[x_fill[:frame+1], y_fill[:frame+1]])
        return scatter,

    ani = animation.FuncAnimation(fig, update, frames=len(x_fill), interval=speed,
blit=True, repeat=False)
    canvas.draw()
```

Anima la trayectoria de los puntos de relleno.

- Limpia la gráfica (ax.clear()).
- Dibuja el contorno de la figura en color azul (ax.plot()).
- Crea un objeto de dispersión (scatter) para los puntos de relleno.
- Valida la velocidad de animación con validate_speed().
- Función interna update(frame):
    - Actualiza la posición de los puntos hasta el paso actual (frame).
    - Se usa scatter.set_offsets() para modificar las coordenadas de los puntos.
- Inicia la animación (FuncAnimation) con la cantidad de pasos necesarios.
- Redibuja la gráfica (canvas.draw()) en la ventana de Tkinter.

Función start_ui()

```
def start_ui():

    """Inicializa la interfaz gráfica de la aplicación."""

    show_loading_screen()

    global resolution_entry, speed_entry, svg_var, canvas, fig, ax, root,
point_color

    root = tk.Tk()
    root.title("Grupo 3 - Impresora 3D")
    root.configure(bg='black')
    root.option_add('*TButton*background', 'crimson')
    root.option_add('*TButton*foreground', 'white')
    root.option_add('*TLabel*foreground', 'white')
    root.option_add('*TEntry*foreground', 'black')
    root.option_add('*TEntry*background', 'white')
    root.option_add('*TFrame*background', 'black')
    root.option_add('*OptionMenu*foreground', 'white')
    root.option_add('*OptionMenu*background', 'crimson')
    root.option_add('*Menu*foreground', 'white')
    root.option_add('*Menu*background', 'crimson')
    root.configure(highlightbackground='black', highlightcolor='black',
highlightthickness=2)

    control_frame = tk.Frame(root, bg='black')
    control_frame.pack(side=tk.LEFT, padx=10, pady=10)

    tk.Label(control_frame, text="Seleccionar archivo SVG", bg='black',
fg='white').pack()
    svg_var = tk.StringVar(value="forma1.svg")
    svg_options = ["avion.svg", "casa.svg", "forma1.svg", "forma2.svg"]
    svg_menu = tk.OptionMenu(control_frame, svg_var, *svg_options)
    svg_menu.config(bg='crimson', fg='white')
    svg_menu.pack()
```

```
    tk.Label(control_frame, text="Distancia entre puntos (0.1 - 0.5 mm)",
bg='black', fg='white').pack()
    resolution_entry = tk.Entry(control_frame)
    resolution_entry.pack()
    resolution_entry.insert(0, "0.1")

    tk.Label(control_frame, text="Velocidad de animación (ms)", bg='black',
fg='white').pack()
    speed_entry = tk.Entry(control_frame)
    speed_entry.pack()
    speed_entry.insert(0, "10")

    color_button = tk.Button(control_frame, text="Seleccionar Color",
command=choose_color, bg='crimson', fg='white')
    color_button.pack()

    load_button = tk.Button(control_frame, text="Cargar SVG", command=load_svg,
bg='crimson', fg='white')
    load_button.pack()

    fig, ax = plt.subplots()
    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

    point_color = 'red'
    root.mainloop()
```

Inicializa la interfaz gráfica de la aplicación.

- Muestra la pantalla de carga con show_loading_screen().
- Crea la ventana principal (root) con título y fondo negro.
- Configura estilos globales para botones, etiquetas y entradas (option_add()).
- Crea el panel de control (control_frame) en el lado izquierdo:
    - Selector de archivos SVG (OptionMenu) con valores predefinidos.
    - Campo de entrada para la distancia entre puntos (Entry).
    - Campo de entrada para la velocidad de animación (Entry).
    - Botón para seleccionar color (Button).
    - Botón para cargar el archivo SVG (Button).
- Crea la figura de Matplotlib (fig, ax) para mostrar la animación.
- Integra la gráfica en Tkinter con FigureCanvasTkAgg().
- Inicia el bucle principal de la aplicación con root.mainloop().

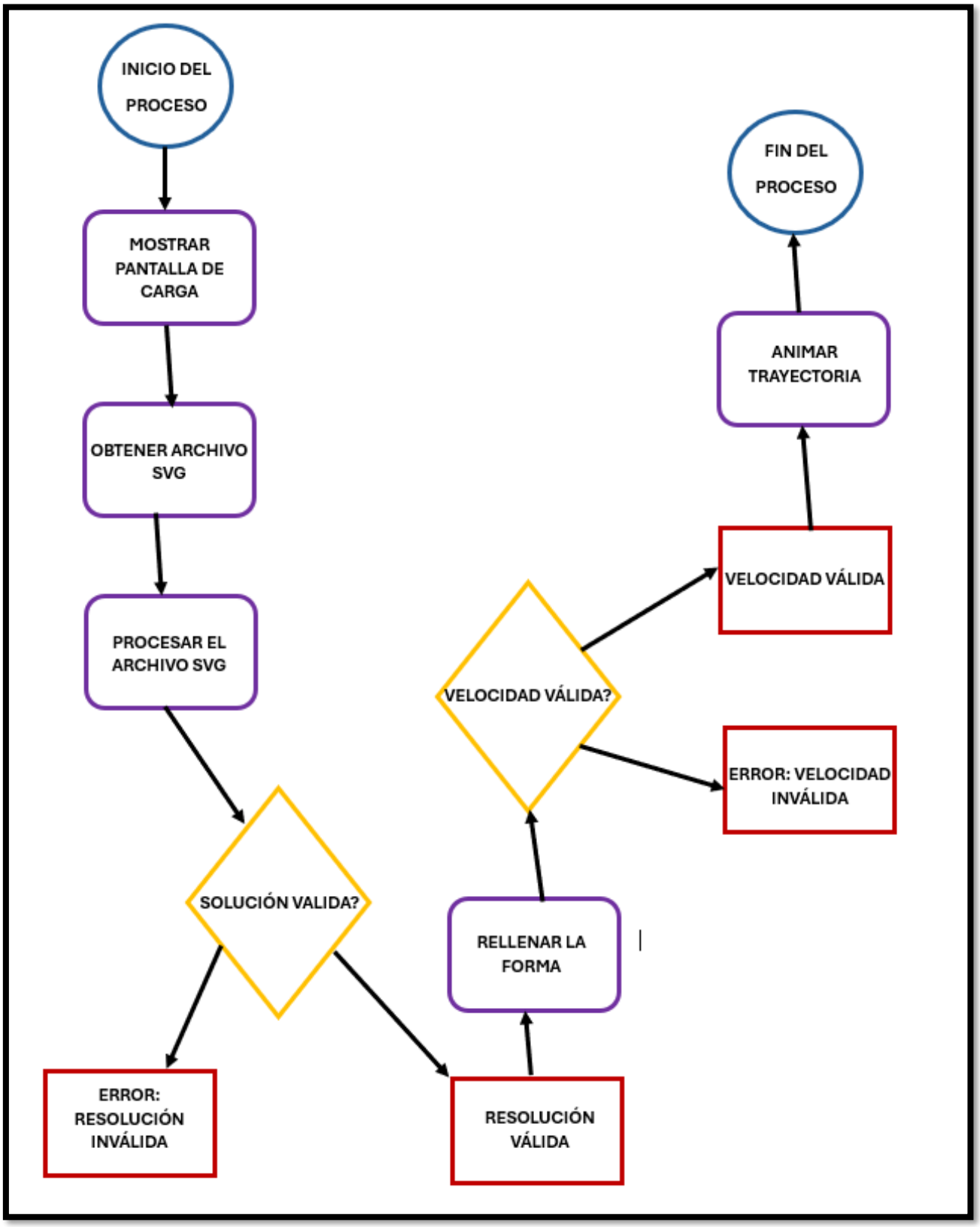
Bloque if __name__ == "__main__":

```
if __name__ == "__main__":
    start_ui()
```

Verifica que el script se esté ejecutando como programa principal.

- Llama a start_ui() para iniciar la aplicación.

- **Diagrama de flujo**



- **Pseudocódigo**

Para realizar el Pseudocódigo utilizaremos la herramienta PSeint el cuál es un intérprete de Pseudocódigo.

```
Proceso Impresora3D
    // Definir las variables
    Definir a Como Real
    Definir b Como Real
    Definir opcion Como Entero
    Definir archivo Como Cadena
    Definir resolucion Como Real
    Definir velocidad Como Entero

    // Simulación de carga
    Escribir "Cargando..."
    Para i ← 1 Hasta 100 Hacer
        Escribir "Cargando " + ConvertirATexto(i) + "%"
        Esperar 0.02 Segundos
    FinPara

    // Selección de archivo SVG
    Escribir "Seleccione un archivo SVG (1. avion.svg, 2. casa.svg, 3. forma1.svg, 4. forma2.svg)
    Leer opcion
    Segun opcion Hacer
        1: archivo ← "avion.svg"
        2: archivo ← "casa.svg"
        3: archivo ← "forma1.svg"
        4: archivo ← "forma2.svg"
        De Otro Modo:
            Escribir "Opción inválida. Seleccionando por defecto forma1.svg"
            archivo ← "forma1.svg"
    FinSegun
```

```

    // Entrada de datos de resolución y velocidad
    Repetir
        Escribir "Ingrese la distancia entre puntos (0.1 - 0.5 mm):"
        Leer resolucion
    Hasta Que resolucion ≥ 0.1 Y resolucion ≤ 0.5

    Repetir
        Escribir "Ingrese la velocidad de animación (ms, valor positivo):"
        Leer velocidad
    Hasta Que velocidad > 0

    // Simulación del llenado de figura
    Escribir "Procesando archivo: " + archivo
    Escribir "Generando puntos de llenado..."

    // Simulación de animación del llenado
    Para a ← 0 Hasta 10 Con Paso resolucion Hacer
        Para b ← -1 Hasta 8 Con Paso resolucion Hacer
            Escribir "Dibujando punto en: " + ConvertirATexto(a) + ", " + ConvertirATexto(b)
            Esperar velocidad Milisegundos
        FinPara
    FinPara

    Escribir "Animación completada."
FinProceso
```

- **Detalles importantes de la implementación**

La implementación del código sigue una estructura modular, dividiendo cada funcionalidad en funciones específicas para tareas como la carga del archivo SVG, la validación de parámetros, el procesamiento de la trayectoria y la animación. Esta organización facilita la modificación y depuración sin afectar otras partes del programa, permitiendo un mantenimiento eficiente y escalable.

Uno de los aspectos fundamentales es el manejo de archivos SVG, ya que la aplicación emplea la biblioteca svgpathtools para interpretar y extraer sus trayectorias. Es crucial asegurarse de que estos archivos contengan trayectorias bien definidas y sin elementos innecesarios que puedan generar errores en el procesamiento. Un diseño limpio y estructurado del archivo SVG mejora la precisión de la simulación.

La interfaz gráfica se desarrolla utilizando Tkinter, proporcionando controles intuitivos para la selección de archivos, ajuste de parámetros y ejecución de la animación. Es importante revisar la disposición y el diseño de los widgets para garantizar una experiencia de usuario fluida y sin interrupciones, permitiendo un uso eficiente y comprensible del programa.

Para evitar fallos en la simulación, la aplicación incorpora un sistema de validación de parámetros del usuario. Se establecen límites en valores clave, como la resolución de impresión (entre 0.1 y 0.5 mm) y la velocidad, que debe ser un número positivo. El cumplimiento de estas restricciones es esencial para que la animación se ejecute correctamente y represente fielmente la trayectoria de impresión.

La animación de la trayectoria de impresión se logra mediante Matplotlib.animation, lo que permite visualizar en tiempo real el proceso de impresión simulado. Es necesario configurar adecuadamente los límites de los ejes y la velocidad de animación para que la representación sea clara, fluida y fácil de interpretar. Además, la selección de colores para la visualización juega un papel importante, ya que se ofrece la opción de cambiar el color de los puntos de relleno a través de colorchooser, por lo que se recomienda elegir colores que contrasten con el fondo y el contorno para mejorar la legibilidad del trazado.

Finalmente, la aplicación incluye una pantalla de carga inicial con una barra de progreso simulada, mejorando la experiencia del usuario mientras el programa se prepara para

ejecutarse. También es importante considerar la compatibilidad y los requisitos del software, ya que la ejecución del código depende de la correcta instalación de las bibliotecas Tkinter, numpy, matplotlib y svgpathtools. Antes de ejecutar el programa, se recomienda verificar que todas estas dependencias estén correctamente instaladas en el entorno de Python.

VI. RESULTADOS

La implementación del sistema de simulación de impresión 3D en Python ha dado como resultado una herramienta funcional que permite visualizar el proceso de impresión a partir de archivos SVG. La aplicación carga y procesa correctamente los archivos vectoriales, extrayendo sus trayectorias y representándolas gráficamente mediante Matplotlib. Se ha logrado que el contorno del objeto sea identificado con precisión y que los patrones de relleno se generen de manera adecuada, siguiendo una disposición en zigzag.

Uno de los principales logros es la capacidad del programa para ajustar la resolución del relleno, permitiendo que el usuario seleccione la distancia entre puntos en un rango entre 0.1 mm y 0.5 mm. Esta funcionalidad garantiza que el patrón generado sea lo suficientemente denso o disperso según la necesidad del usuario, simulando diferentes configuraciones de una impresora 3D real. Además, la validación de estos valores evita que se ingresen parámetros erróneos que puedan afectar la simulación.

La animación del proceso de impresión ha sido otro resultado significativo, permitiendo visualizar en tiempo real cómo se iría completando la figura capa por capa. La inclusión de controles de velocidad ha mejorado la experiencia del usuario, ya que puede ajustar la rapidez con la que los puntos de impresión aparecen en la pantalla. La implementación de colores personalizables para los puntos de relleno es otro avance, brindando flexibilidad para diferenciar diversas áreas del modelo impreso.

Los resultados del desarrollo matemático del sistema de simulación de impresión 3D se centran en la generación de mallas 2D, la verificación de puntos dentro de un polígono y la optimización de la trayectoria de impresión. Estos aspectos matemáticos han permitido una representación precisa de la distribución de puntos en el espacio de impresión y una eficiente simulación del proceso de impresión capa por capa.

En la creación de la malla, se ha logrado generar una cuadrícula uniforme de puntos a partir de las ecuaciones:

$$x_i = x_{min} + i * d_x$$

$$y_j = y_{min} + j * d_y$$

Estos cálculos aseguran que los puntos estén distribuidos con un espaciado constante dxd_xdx y dyd_ydy, lo que permite controlar la resolución de la impresión. Se verificó que la generación de la malla sea consistente para diferentes tamaños de figuras, asegurando que los puntos cubran uniformemente el área definida por el modelo a imprimir.

La verificación de puntos dentro de un polígono se implementó utilizando el criterio basado en cruces de segmentos con un rayo horizontal, expresado como:

$$C(x,y) = \sum_{i=1}^{n} I((x\_i, y\_i), (x_{i+1}, y_{i+1}), (x,y))$$

Donde la función I(A,B,P)I(A,B,P)I(A,B,P) determina si un segmento cruza el rayo horizontal desde el punto PPP. A través de este método, se ha conseguido una clasificación precisa de los puntos que deben formar parte del relleno, evitando errores en la representación de la figura dentro de la malla de impresión. Las pruebas realizadas han mostrado que esta verificación es efectiva para figuras de diferente complejidad, incluyendo polígonos convexos y cóncavos.

En cuanto a la optimización de la trayectoria, se han aplicado reglas de desplazamiento dentro de la malla para minimizar movimientos innecesarios, mejorando la eficiencia del recorrido. La estrategia implementada sigue un patrón de desplazamiento continuo en columnas de la malla:

- Ascenso en una columna: (x,y)→(x,y+dy)
- Descenso en la siguiente columna: (x+dx,y)→(x+dx,y-dy)

Este patrón garantiza que la simulación del proceso de impresión sea fluida y que la trayectoria minimice los saltos entre puntos, emulando de manera más realista el comportamiento de una impresora 3D real.

Por lo tanto, el análisis matemático y la implementación de estos conceptos han permitido que la simulación se base en modelos geométricos precisos, asegurando una adecuada representación de los objetos a imprimir. Además, la correcta parametrización de la malla y la verificación de puntos han optimizado la distribución del relleno y la eficiencia del recorrido, logrando un sistema robusto y adaptable a diferentes configuraciones de impresión.

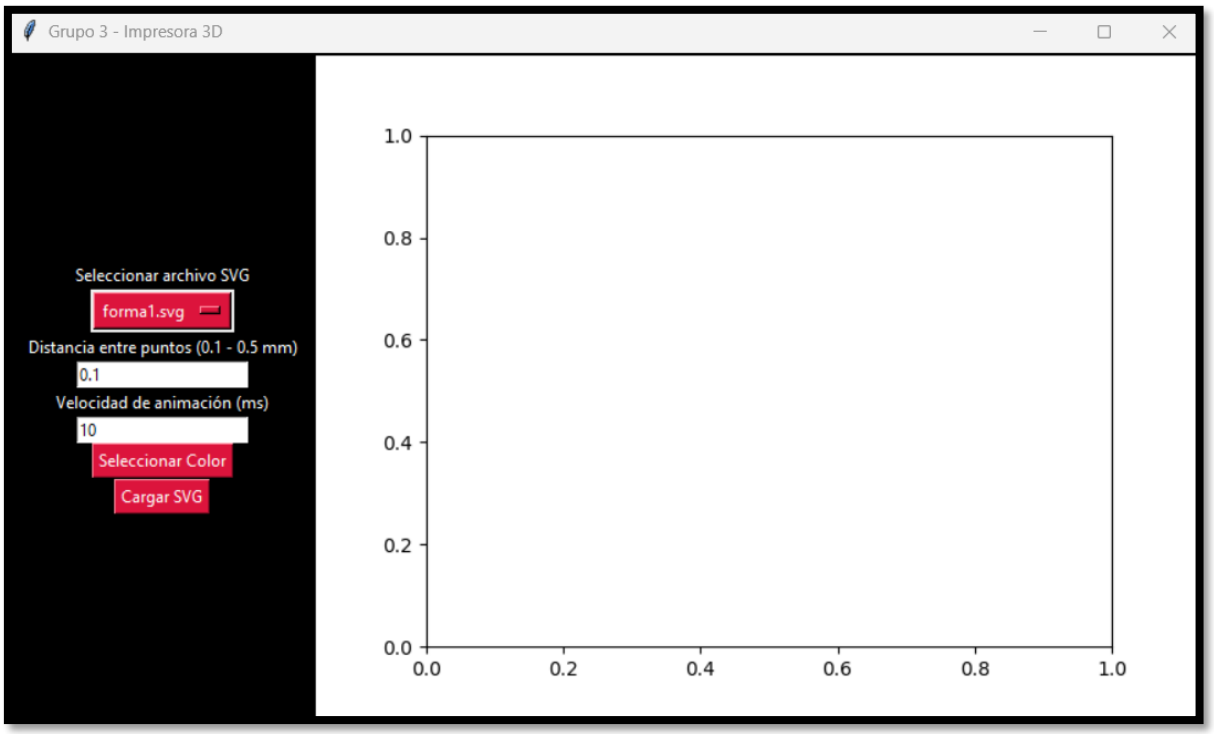
VII. CONCLUSIONES

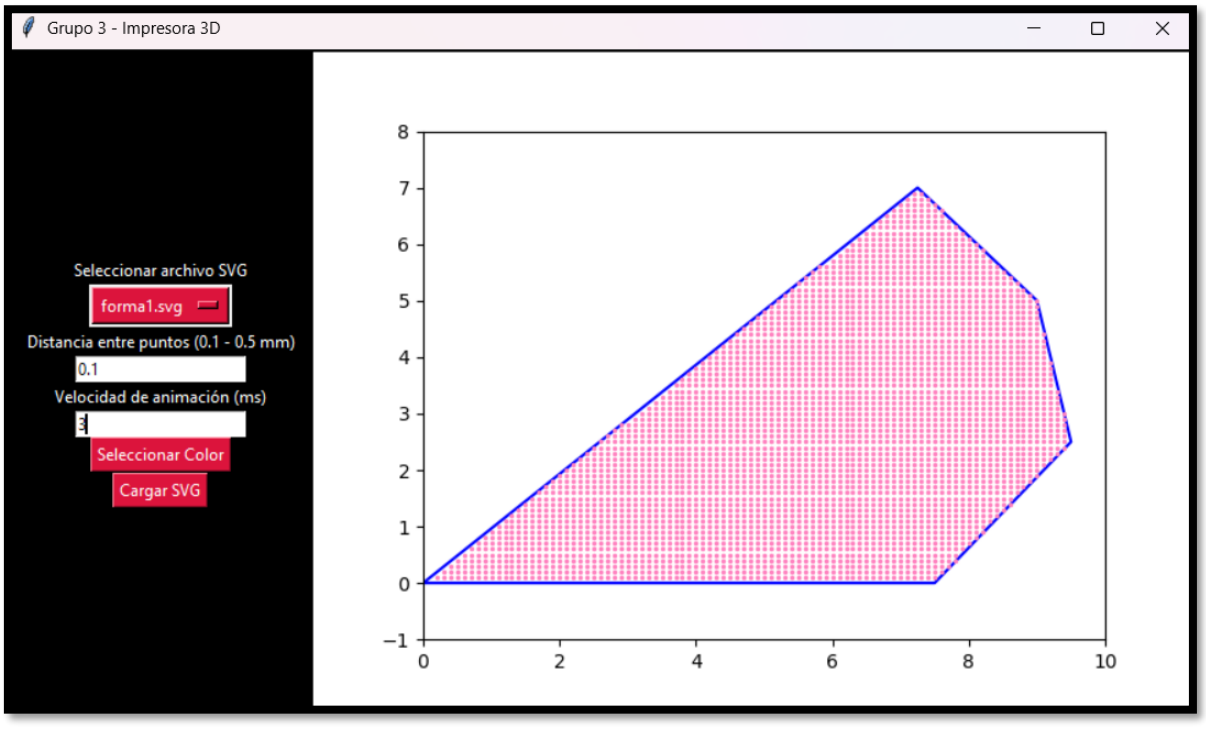
- En conclusión, l**a aplicación permite visualizar y simular la trayectoria de puntos dentro de una figura SVG de manera interactiva, que a su vez simula el recorrido de una impresora 3D.**
- El uso de numpy y svgpathtools facilita la manipulación de los datos extraídos del SVG, permitiendo un procesamiento eficiente de los contornos y la generación de puntos de relleno **y animación de manera estructurada.**
- El código está estructurad**o con funciones independientes para cada tarea específica, y a su vez están documentadas, lo que mejora su legibilidad, mantenimiento y escalabilidad. Esto permite futuras mejoras o modificaciones sin afectar la funcionalidad central.**
- El software permite modificar parámetros clave, como la resolución de impresión y la velocidad de animación, lo que ayuda a evaluar diferentes configuraciones antes de una impresión real.

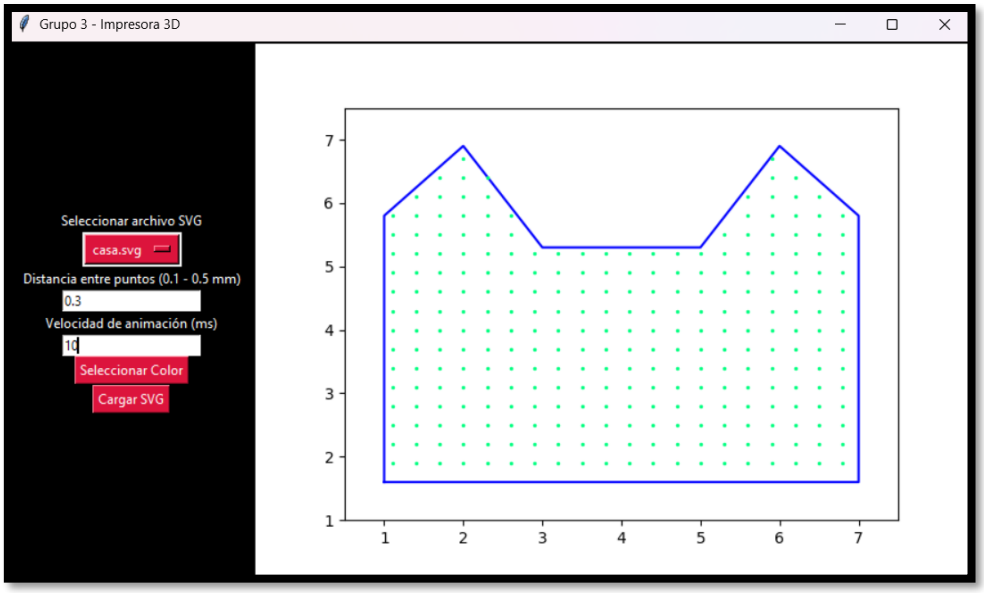
VIII. RECOMENDACIONES

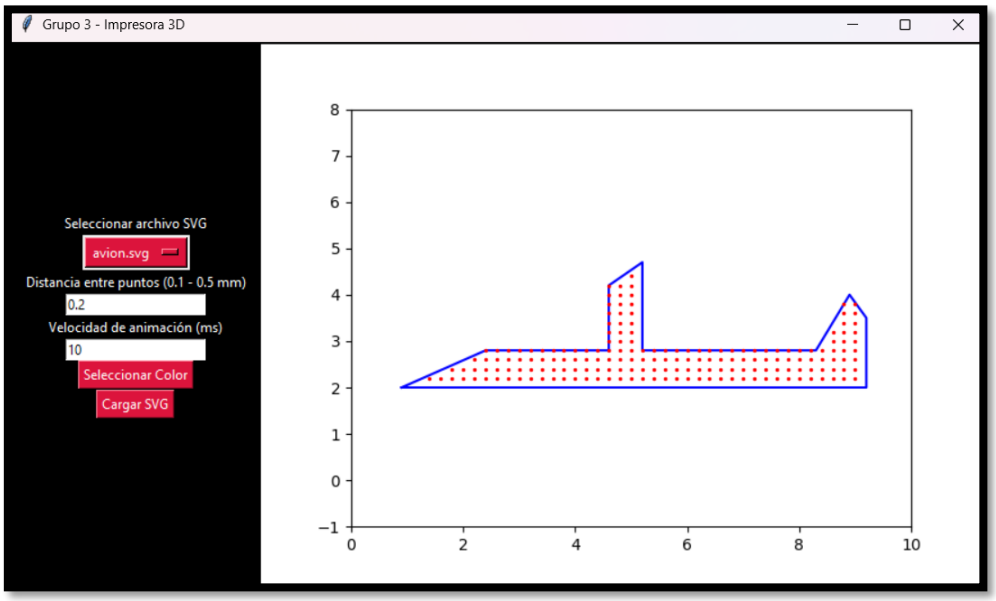
- Se recomienda utilizar archivos SVG que contengan trayectorias bien definidas y sin elementos innecesarios que puedan afectar la simulación. Archivos con demasiados detalles o elementos como rellenos complejos pueden generar errores en la interpretación del contorno y dificultar el correcto funcionamiento del código.
- Se recomienda que la distancia entre puntos debe estar entre 0.1 mm y 0.5 mm para garantizar un equilibrio entre precisión y eficiencia. Si el valor es demasiado bajo, la simulación puede volverse muy densa y lenta, mientras **que, si es demasiado alto, podría perderse información relevante en la trayectoria de impresión.**
- Se recomienda seleccionar colores de alto contraste para los puntos de relleno y el contorno, con el fin de mejorar la visualización de la trayectoria de impresión. Esto facilita la interpretación de los datos y ayuda a identificar posibles errores en la simulación.
- Dado que la animación puede consumir una cantidad considerable de procesamiento, se sugiere probar la aplicación en diferentes dispositivos. Si se detecta un rendimiento lento, se pueden ajustar los parámetros de resolución y velocidad para optimizar la ejecución.

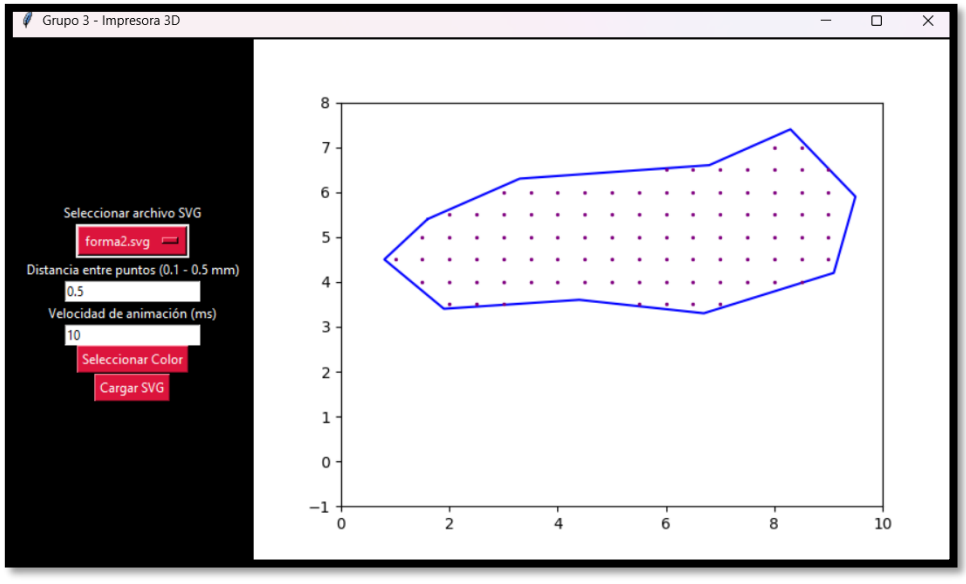
IX. ANEXOS


Grupo 3 - Impresora 3D
Seleccionar archivo SVG
forma1.svg
Distancia entre puntos (0.1 - 0.5 mm)
0.1
Velocidad de animación (ms)
10
Seleccionar Color
Cargar SVG
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Grupo 3 - Impresora 3D
Seleccionar archivo SVG
forma1.svg
Distancia entre puntos (0.1 - 0.5 mm)
0.1
Velocidad de animación (ms)
3
Seleccionar Color
Cargar SVG
8
7
6
5
4
3
2
1
0
−1
0
2
4
6
8
10


Grupo 3 - Impresora 3D
Seleccionar archivo SVG
casa.svg
Distancia entre puntos (0.1 - 0.5 mm)
0.3
Velocidad de animación (ms)
10
Seleccionar Color
Cargar SVG


Grupo 3 - Impresora 3D
Seleccionar archivo SVG
avion.svg
Distancia entre puntos (0.1 - 0.5 mm)
0.2
Velocidad de animación (ms)
10
Seleccionar Color
Cargar SVG


Grupo 3 - Impresora 3D
Seleccionar archivo SVG
forma2.svg
Distancia entre puntos (0.1 - 0.5 mm)
0.5
Velocidad de animación (ms)
10
Seleccionar Color
Cargar SVG

X. LINKS

- **LINK DEL REPOSITORIO**

https://github.com/LAINE30/Proyecto-Final-IIB

- **LINK DEL VIDEO**

https://github.com/LAINE30/Proyecto-Final-IIB/blob/main/assets/Proyecto-Impresora.mp4

- **LINK DE LAS DIAPOSITIVAS**

https://github.com/LAINE30/Proyecto-Final-IIB/blob/main/Impresora%203D%20-%20Grupo%233.pdf

XI. REFERENCIAS

[I] Aula. (2023, 23 mayo). Qué es la impresión 3D, importancia y cómo funciona la fabricación aditiva. *aula21 | Formación para la Industria*. https://www.cursosaula21.com/que-es-la-impresion-3d/

[II] [1] Wikipedia contributors, “Point in polygon,” Wikipedia, The Free Encyclopedia, 07-Feb-2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Point_in_polygon&oldid=1274503173.

[III] [2] “12.1: Conceptos geométricos básicos y figuras,” Mathematics LibreTexts, 10-May-2023. [Online]. Available: https://math.libretexts.org/Courses/Santiago_Canyon_College/HiSet_Mathematica_(Lopez)/12%3A_Geometria/12.01%3A_Conceptos_geometricos_basicos_y_figuras. [Accessed: 12-Feb-2025].

[IV] [3] E. P. J. Ver Todas las Entradas de, “Punto en polígono,” El blog del profe José, 08-Jul-2015. [Online]. Available: https://elprofejose.com/2015/07/08/punto-en-poligono/. [Accessed: 12-Feb-2025].

[V] [4] Cadence.com. [Online]. Available: https://www.cadence.com/en_US/home/explore/what-is-meshing.html#:~:text=Meshing%20or%20mesh%20generation%20discretizes,elements%20using%20partial%20differential%20equations. [Accessed: 12-Feb-2025].