Escuela Politécnica Nacional

Facultad de Ingeniería en Sistemas

Métodos Numéricos (Gr1cc)

Informe del Proyecto IIB

Simulación de una Impresora 3D

Integrantes:

- David Pilataxi
- Freddy Jiménez
- Mijael Molina

PROFESOR: Jonathan Alejandro Zea G.
FECHA DE ENTREGA: 12/02/2025

Índice

1. Objetivos

1.1. Objetivo General

Desarrollar una simulación de impresora 3D que permita representar el proceso de impresión a partir de un archivo SVG. La simulación incluye la carga de un modelo 2D, la generación de una trayectoria de impresión basada en la resolución definida por el usuario, y la visualización del proceso de impresión en una interfaz gráfica.

1.2. Objetivos Específicos

- Crear una interfaz gráfica que permita al usuario cargar modelos SVG, definir parámetros de impresión (resolución y velocidad) y visualizar la simulación en tiempo real.
- Desarrollar una función que permita cargar archivos SVG y extraer los puntos que definen los polígonos del modelo.
- Implementar un algoritmo que genere una trayectoria de impresión en forma de zigzag, considerando la resolución definida por el usuario (horizontal y vertical).
- Desarrollar una simulación visual que muestre el progreso de la impresión en tiempo real para graficar la trayectoria.

2. Introducción

Las impresoras 3D han revolucionado la manufactura y el diseño, permitiendo la creación de objetos tridimensionales a partir de modelos digitales. Estas máquinas funcionan depositando material capa por capa, siguiendo una trayectoria precisa que define la forma del objeto. Sin embargo, comprender y simular este proceso es fundamental para optimizar la impresión y garantizar resultados de alta calidad.

En este escenario, se solicitó desarrollar una simulación de una impresora 3D que permita imprimir objetos bidimensionales a partir de archivos SVG. El proyecto consiste en crear una interfaz gráfica que permita al usuario cargar modelos SVG, configurar parámetros de impresión (como la resolución y la velocidad) y visualizar la trayectoria que seguiría la impresora para replicar el modelo. Esta simulación no solo servirá como una herramienta educativa para entender el funcionamiento de las impresoras 3D, sino también como un prototipo para probar diferentes configuraciones antes de ejecutar una impresión real.

El problema implica el procesamiento de archivos SVG para extraer los contornos del modelo, la generación de una trayectoria de impresión basada en la resolución definida por el usuario, y la visualización de esta trayectoria en tiempo real. Para lograrlo, se utilizaron herramientas como tkinter para la interfaz gráfica, matplotlib para la visualización de la trayectoria, y técnicas de geometría computacional para garantizar que los puntos de la trayectoria se encuentren dentro del área del modelo.

Finalmente, la solución fue implementada en Python, aprovechando las bibliotecas disponibles para realizar cálculos, procesar archivos SVG y generar simulaciones interactivas. Este proyecto no solo cumple con los requisitos solicitados, sino que también sienta las bases para futuras mejoras, como la extensión a modelos tridimensionales o la incorporación de algoritmos más avanzados para la optimización de trayectorias.

3. Desarrollo

3.1. Descripción de la solución

La solución para simular una impresión 3D se basa en tres pilares principales:

1. **Procesamiento del SVG**: Extraer los puntos que definen el polígono desde el archivo SVG.

2. **Generación de la trayectoria**: Calcular los puntos que la impresora debe seguir para cubrir el área del polígono.

3. **Simulación visual:** Mostrar la trayectoria en tiempo real utilizando gráficos.

Cada uno de estos pasos involucra un desarrollo matemático y algorítmico específico, que detallaremos a continuación.

3.2. Desarrollo matemático

Procesamiento del SVG

El archivo SVG contiene la definición del polígono que representa el objeto a imprimir. Matemáticamente, un polígono es una figura plana delimitada por una secuencia de puntos (vértices) conectados por segmentos de línea. Para que el polígono esté correctamente definido, el primer y último punto deben coincidir, cerrando así la figura. Si no es así, se agrega manualmente el primer punto al final de la lista. Esta propiedad es fundamental en geometría, ya que un polígono cerrado define un área finita.

El procesamiento del SVG implica extraer estos puntos y almacenarlos en una lista de coordenadas (x, y).

Generación de la Trayectoria

La generación de la trayectoria es el núcleo matemático del proyecto. Aquí, el objetivo es calcular los puntos que la impresora debe seguir para cubrir el área del polígono de manera eficiente. Para ello, se utilizan conceptos de geometría analítica y algoritmos de relleno de polígonos.

- **Límites del polígono**

Primero, se calculan los valores mínimos y máximos en los ejes X e Y (min_x, max_x, min_y, max_y). Estos valores definen un rectángulo que contiene completamente el polígono:

- min_x = min(x1, x2, ..., xn)
- max_x = max(x1, x2, ..., xn)
- min_y = min(y1, y2, ..., yn)
- max_y = max(y1, y2, ..., yn)

Estos límites permiten definir un área rectangular sobre la cual se generará la trayectoria.

- **Resolución de impresión**

La resolución (res_x, res_y) define la distancia entre los puntos de la trayectoria. Esto se traduce en un espaciado uniforme en los ejes X e Y. Por ejemplo, si res_x = 1 y res_y = 1, la impresora se moverá en incrementos de 1 unidad en ambas direcciones.

- **Patrón de zigzag**

La trayectoria se genera en un patrón de zigzag, alternando la dirección en cada columna. Esto significa que en una columna, la impresora se mueve de abajo hacia arriba, y en la siguiente, de arriba hacia abajo. Matemáticamente, esto se logra invirtiendo el orden de los valores de Y en cada columna:

- Si la dirección es hacia arriba:

y = min_y, min_y + res_y, ..., max_y

- Si la dirección es hacia abajo:

y = max_y, max_y res_y, ..., min_y

Este patrón optimiza el tiempo de impresión, ya que reduce los movimientos innecesarios de la impresora.

- **Verificación de puntos dentro del polígono**

Para cada punto (x, y) generado, se verifica si está dentro del polígono utilizando el algoritmo del punto en polígono (Ray Casting). Este algoritmo cuenta cuántas veces un rayo horizontal que parte del punto cruza los bordes del polígono. Si el número de cruces es impar, el punto está dentro del polígono; si es par, está fuera.

- **Verificación de puntos en un segmento del polígono**

Se determina si un punto está sobre un segmento del polígono proyectándolo ortogonalmente sobre la línea que une dos vértices consecutivos. Se calcula un parámetro t mediante la fórmula.

$$t = \frac{(p - p_1) \cdot (p_2 - p_1)}{|p_2 - p_1|^2}$$

donde p1 y p2 son los extremos del segmento y p es el punto a evaluar. Si t se encuentra entre 0 y 1, la proyección cae dentro del segmento; además, se verifica que la distancia entre p y su proyección sea menor que una tolerancia establecida para considerar que el punto está efectivamente sobre el segmento.

Simulación Visual: Interpolación y Tiempo Real

La simulación visual muestra cómo la impresora sigue la trayectoria en tiempo real. Los pasos son:

1. Interpolación lineal:

- Dados dos puntos consecutivos (x1, y1) y (x2, y2), la posición intermedia se calcula como:
  - x(t) = x1 + t * (x2 - x1)
  - y(t) = y1 + t * |(y2 - y1)

Donde t es un valor entre 0 y 1.

2. Actualización en tiempo real:

La simulación se actualiza en intervalos de tiempo Δt, controlados por la velocidad de impresión. Un valor alto de velocidad reduce Δt, haciendo la simulación más rápida.

3. Dibujo de la trayectoria:

Los puntos de la trayectoria se dibujan en un plano 2D usando la biblioteca matplotlib. La posición actual de la impresora se muestra como un punto en movimiento.

4. Control de Velocidad

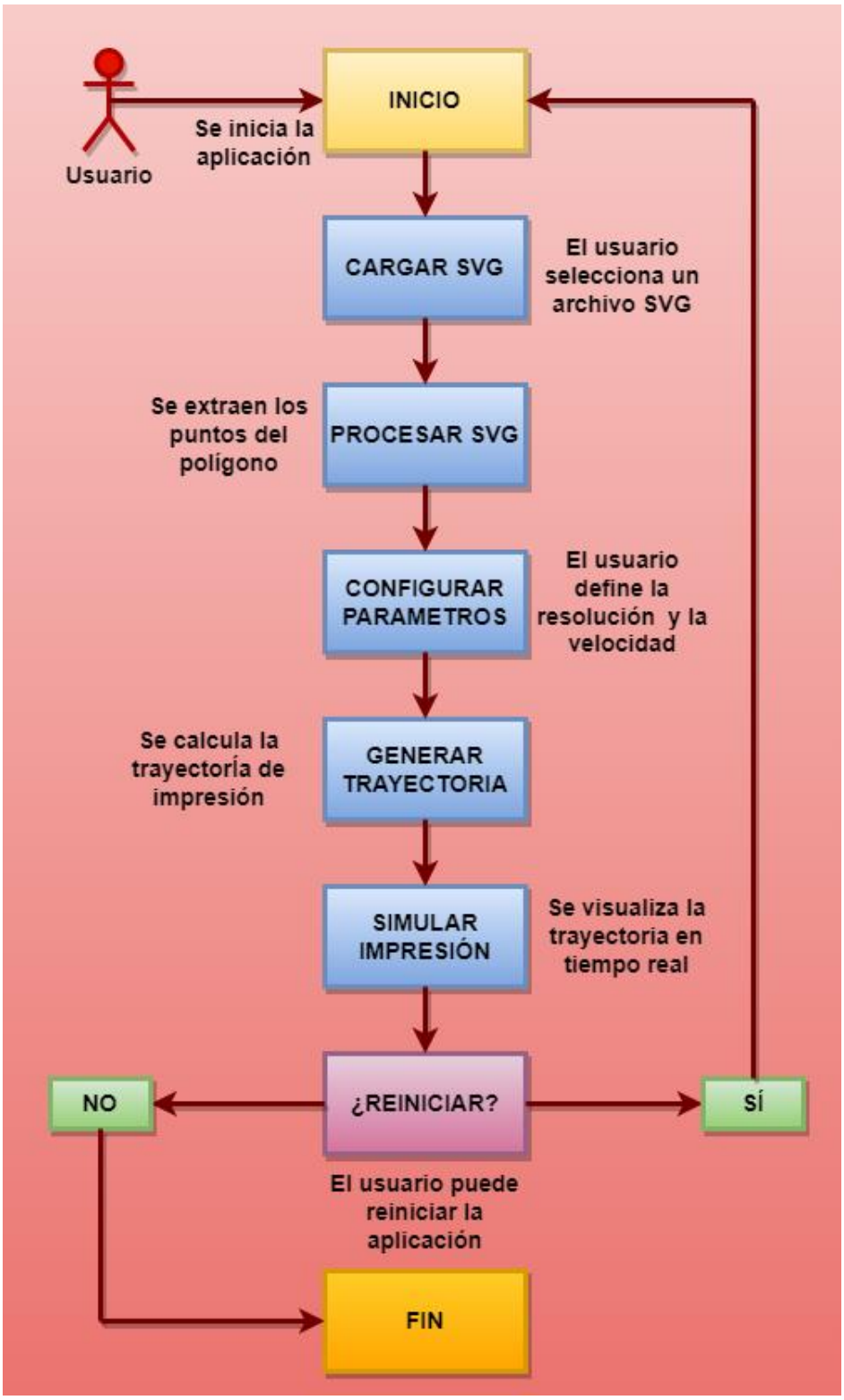
La velocidad de la simulación se ajusta cambiando el intervalo de tiempo Δt entre actualizaciones. Matemáticamente:

- Δt = 1 / v’

Donde v es la velocidad de impresión (puntos por segundo).

3.3. Diagrama de flujo y pseudo código

Diagrama de flujo general



Pseudocódigo para trayectoria "arriba y abajo"

Este pseudocódigo genera una trayectoria que alterna direcciones en cada columna, subiendo y bajando de manera eficiente. Es una mejora sobre el recorrido vertical constante, ya que optimiza

el movimiento de la impresora y es más realista en términos de simulación de impresión 3D.

```
# Inicializar la dirección inicial (False = de abajo hacia arriba, True = de arriba hacia abajo)
direccion_arriba = False

# Recorrer cada columna en X
Para cada x en el rango (min_x, max_x, res_x):

    # Si la dirección es hacia arriba
    Si direccion_arriba es False:
        # Recorrer Y de min_y a max_y (de abajo hacia arriba)
        Para cada y en el rango (min_y, max_y, res_y):
            Si punto_en_poligono(poligono_path, (x, y)):
                Agregar (x, y) a la trayectoria

    # Si la dirección es hacia abajo
    Sino:
        # Recorrer Y de max_y a min_y (de arriba hacia abajo)
        Para cada y en el rango (max_y, min_y, -res_y):
            Si punto_en_poligono(poligono_path, (x, y)):
                Agregar (x, y) a la trayectoria

    # Alternar la dirección para la siguiente columna
    direccion_arriba = not direccion_arriba
```

3.4. Implementación en Python

Procesamiento de SVG

Para procesar los archivos SVG, se utiliza la biblioteca xml.etree.ElementTree, que permite leer y manipular archivos XML, como es el caso de los SVG. En este proyecto, se extraen los puntos que definen los polígonos del archivo SVG. Estos puntos se almacenan en una lista de coordenadas (x, y), que representan los vértices del polígono.

Un detalle crucial es asegurarse de que el polígono esté cerrado. Esto significa que el primer y último punto deben coincidir. Si no es así, se agrega manualmente el primer punto al final de la lista. Esto es importante porque, en gráficos vectoriales, un polígono cerrado es necesario para definir correctamente el área que se va a rellenar durante la impresión.

```python
if self.puntos:
    # Asegurarse de que el polígono esté cerrado
    if self.puntos[0] != self.puntos[-1]:
        self.puntos.append(self.puntos[0])
    self.poligono_path = Path(self.puntos)
    self.dibujar_modelo()
```

Generación de trayectoria

La generación de la trayectoria es uno de los aspectos más importantes del proyecto. Primero, se calculan los límites del polígono, es decir, los valores mínimos y máximos en los ejes X e Y. Estos límites definen el área rectangular que contiene el polígono y sirven como referencia para generar los puntos de la trayectoria.

```python
# Calcula los límites del polígono
min_x = min(p[0] for p in self.puntos)
max_x = max(p[0] for p in self.puntos)
min_y = min(p[1] for p in self.puntos)
max_y = max(p[1] for p in self.puntos)

self.trayectoria = []  # Inicializa la lista de trayectoria
reverse = False  # Controla la dirección del recorrido (zigzag)
```

Luego, se genera la trayectoria en un patrón de zigzag. Esto implica alternar la dirección del recorrido en cada fila: en una columna, la impresora se mueve de abajo hacia arriba, y en la siguiente, de arriba hacia abajo. Este enfoque optimiza el tiempo de impresión, ya que reduce los movimientos innecesarios de la impresora.

```python
# Genera la trayectoria en zigzag
for x in np.arange(min_x, max_x + res_x, res_x):  # Recorre el eje X
    y_values = np.arange(min_y, max_y + res_y, res_y)  # Recorre el eje Y
    if reverse:
        y_values = y_values[::-1]  # Invierte el orden de Y para el zigzag

    for y in y_values:
        punto = (x, y)
        # Verifica si el punto está dentro del polígono o en su contorno
        if punto_en_poligono(self.poligono_path, punto):
            self.trayectoria.append(punto)  # Agrega el punto a la trayectoria

    reverse = not reverse  # Alterna la dirección para la siguiente columna
```

Cada punto de la trayectoria se verifica para asegurarse de que esté dentro del polígono o en su contorno. Esto se hace utilizando la función punto_en_poligono, que combina técnicas de geometría computacional para determinar si un punto está dentro del área del polígono. Si el punto no está dentro del polígono, se descarta y no se incluye en la trayectoria.

```python
def punto_en_poligono(path, punto, tolerancia=1e-2):
    """
    Verifica si un punto está dentro del polígono o en su contorno
    """
    # Primero verificamos si el punto está dentro del polígono
    if path.contains_point(punto):
        return True

    # Luego verificamos si el punto está en el contorno
    vertices = path.vertices
    for i in range(len(vertices) - 1):
        p1 = vertices[i]
        p2 = vertices[i + 1]
        if punto_en_segmento(p1, p2, punto, tolerancia):
            return True
    return False
```

Simulación visual

La simulación visual se implementa utilizando la biblioteca matplotlib, que permite graficar el modelo y la trayectoria de impresión. El modelo se dibuja como una serie de líneas que conectan los puntos del polígono, mientras que la trayectoria se representa como una secuencia de puntos que se van dibujando en tiempo real.

La simulación se actualiza paso a paso, mostrando el progreso de la impresión. Esto se logra utilizando un bucle que dibuja pequeños segmentos de la trayectoria en intervalos

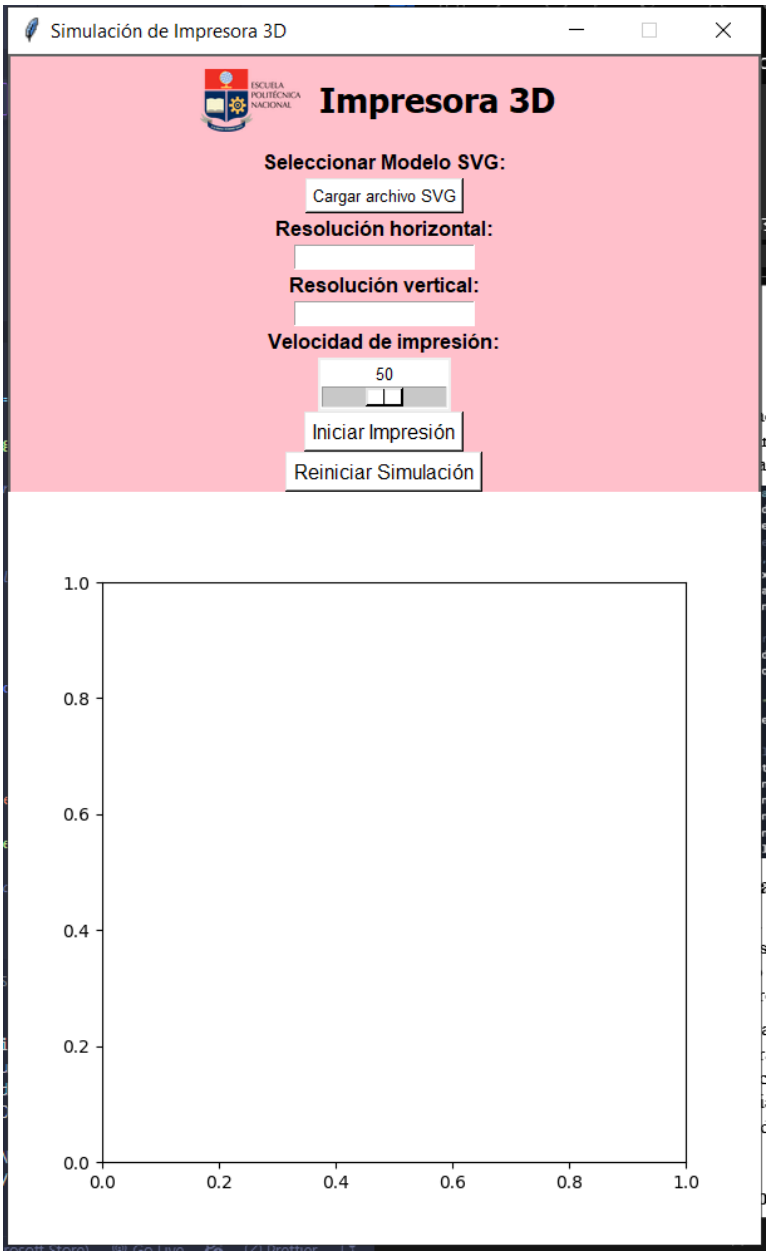
de tiempo controlados por la velocidad de impresión. La velocidad se ajusta mediante un deslizador en la interfaz gráfica, lo que permite al usuario observar la simulación a diferentes velocidades.

```python
def imprimir_paso(self):
    if self.index_imprimir < len(self.trayectoria):  # Verifica si hay más puntos por imprimir
        incremento = self.incremento_puntos  # Número de puntos a dibujar en cada paso
        # Obtiene los puntos de la trayectoria hasta el índice actual
        x_vals, y_vals = zip(*self.trayectoria[:self.index_imprimir + incremento])
        self.ax.plot(x_vals, y_vals, 'r.', markersize=2)  # Dibuja los puntos en rojo
        self.canvas.draw()  # Actualiza el gráfico
        self.index_imprimir += incremento  # Avanza el índice de impresión

        # Controla la velocidad de la simulación
        velocidad = max(1, self.slider_velocidad.get())  # Obtiene la velocidad del deslizador
        self.root.after(int(100 / velocidad), self.imprimir_paso)  # Programa el siguiente paso
    else:
        print("Impresión completada.")
        messagebox.showinfo("Información", "Impresión completada.")

        # Habilita los controles después de completar la impresión
        self.btn_imprimir.config(state=tk.NORMAL)
        for btn in self.botones_modelos:
            btn.config(state=tk.NORMAL)
        self.entry_resolucion_x.config(state=tk.NORMAL)
        self.entry_resolucion_y.config(state=tk.NORMAL)
        self.slider_velocidad.config(state=tk.NORMAL)
```

Interfaz gráfica

La interfaz gráfica se desarrolla utilizando tkinter, una biblioteca estándar de Python para crear aplicaciones de escritorio. La ventana principal incluye varios elementos interactivos, como botones para cargar archivos SVG, campos de entrada para definir la resolución de impresión y un deslizador para ajustar la velocidad.

El área de visualización se implementa utilizando un FigureCanvasTkAgg de matplotlib, que permite integrar gráficos en la interfaz de tkinter. Esto facilita la visualización del modelo y la trayectoria en tiempo real. Además, se incluyen controles para reiniciar la simulación y limpiar el área de visualización, lo que permite al usuario realizar múltiples pruebas sin necesidad de reiniciar la aplicación.



Manejo de errores

El manejo de errores es fundamental para garantizar que la aplicación funcione de manera robusta. Se validan las entradas del usuario, como la resolución de impresión y la velocidad, para asegurarse de que sean valores numéricos válidos. Si el usuario ingresa un valor incorrecto, se muestra un mensaje de error y se detiene la simulación.

Además, se verifica que el archivo SVG cargado sea válido y contenga polígonos. Si el archivo no cumple con estos requisitos, se muestra un mensaje de error y se solicita al usuario que cargue un archivo válido. Esto evita que la aplicación falle debido a entradas incorrectas o archivos mal formados.

```python
def iniciar_impresion(self):
    if not self.puntos or not self.entry_resolucion_x.get() or not self.entry_resolucion_y.get
():
        messagebox.showerror("Error", "Falta cargar un modelo o definir la resolución")
        return

    # Deshabilita los controles durante la impresión
    self.btn_imprimir.config(state=tk.DISABLED)
    for btn in self.botones_modelos:
        btn.config(state=tk.DISABLED)
    self.entry_resolucion_x.config(state=tk.DISABLED)
    self.entry_resolucion_y.config(state=tk.DISABLED)
    self.slider_velocidad.config(state=tk.DISABLED)

    # Inicia la simulación
    self.generar_trayectoria()
    self.index_imprimir = 0
    self.ax.clear()
    self.dibujar_modelo()
    self.ax.set_title("Simulación de Impresión", fontweight="bold")
    self.imprimir_paso()
```

4. Casos de Prueba

4.1. Prueba 1

Archivo: avión.svg.

Resolución horizontal: 0.7

Resolución vertical: 0.7

Velocidad de impresión: 71%

Imagen 2 Sentencia del programa que confirma una figura cerrada

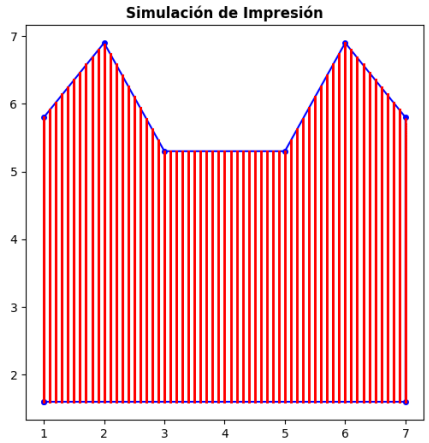
4.2. Prueba 2

Archivo: casa.svg.

Resolución horizontal: 0.1

Resolución vertical: 0.02

Velocidad de impresión: 50%



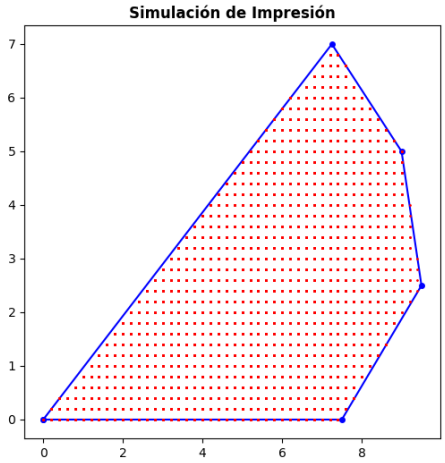
4.3. Prueba 3

Archivo: forma1.svg.

Resolución horizontal: 0.2

Resolución vertical: 0.2

Velocidad de impresión: 100%



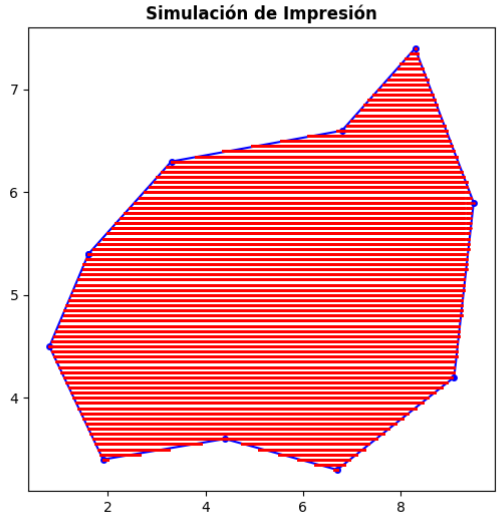
4.4. Prueba 4

Archivo: forma2.svg.

Resolución horizontal: 0.05

Resolución vertical: 0.05

Velocidad de impresión: 100%



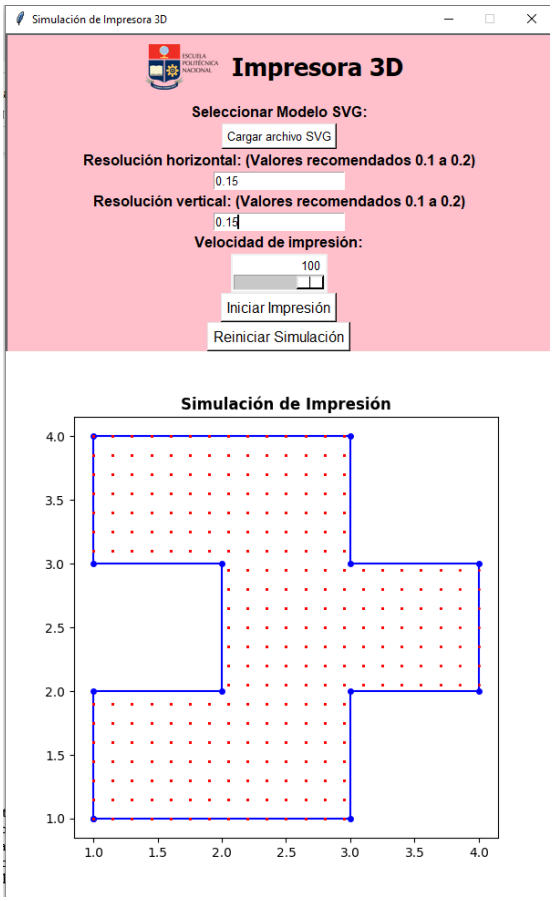
4.5. Prueba 5

Archivo: formatetris.svg.

Resolución horizontal: 0.2

Resolución vertical: 0.2

Velocidad de impresión: 100%



5. Conclusiones

- **Eficacia de la simulación basada en geometría computacional**
  La implementación de algoritmos basados en geometría computacional, como la verificación de puntos dentro de un polígono y la generación de trayectorias en zigzag, ha demostrado ser una herramienta eficaz para simular el proceso de impresión 3D. Estos métodos permiten calcular de manera precisa la trayectoria que debe seguir la impresora, asegurando que se cubra toda el área del modelo sin dejar huecos. Esto refleja la importancia de las técnicas computacionales en la automatización de procesos complejos.
- **Visualización intuitiva del proceso de impresión**
  La integración de gráficos generados con matplotlib proporciona una representación clara e intuitiva del proceso de impresión. La simulación en tiempo real permite observar cómo la impresora recorre la trayectoria, lo que facilita la comprensión del funcionamiento del sistema. Además, la capacidad de ajustar la velocidad de impresión y ver su impacto en la simulación ayuda a identificar posibles optimizaciones en el proceso.
- **Flexibilidad en la configuración de parámetros**
  La capacidad de ajustar parámetros como la resolución de impresión (horizontal y vertical) y la velocidad de impresión hace que la herramienta sea altamente adaptable a diferentes necesidades. Esto permite modelar una amplia variedad de escenarios, desde impresiones de alta precisión hasta simulaciones rápidas para pruebas conceptuales. Esta flexibilidad es especialmente útil para usuarios que buscan optimizar el tiempo y los recursos en sus proyectos.

- **Robustez y manejo de errores**
  La implementación de validaciones y manejo de errores garantiza que la aplicación sea robusta y fácil de usar. Desde la verificación de archivos SVG hasta la validación de entradas numéricas, el sistema está diseñado para evitar fallos y guiar al usuario en caso de errores. Esto no solo mejora la experiencia del usuario, sino que también asegura que los resultados sean confiables y precisos.

Este proyecto ha demostrado que la combinación de técnicas de geometría computacional, visualización gráfica y una interfaz intuitiva puede resultar en una herramienta poderosa para simular procesos complejos, como la impresión 3D. La capacidad de ajustar parámetros y observar los resultados en tiempo real no solo facilita la comprensión del proceso, sino que también abre la puerta a futuras mejoras, como la extensión a modelos tridimensionales o la incorporación de algoritmos más avanzados para la optimización de trayectorias.

6. Preguntas de análisis

¿Qué área del objeto es cubierta por la impresora 3D con su algoritmo?

El algoritmo cubre toda el área interior del polígono definido por el archivo SVG, siempre que se cumplan las siguientes condiciones:

- El polígono esté correctamente cerrado (el primer y último punto coincidan).
- La resolución de impresión (res_x y res_y) sea lo suficientemente pequeña para cubrir toda el área sin dejar huecos.
- Los puntos de la trayectoria estén dentro del polígono o en su contorno, lo cual se verifica mediante la función punto_en_poligono.

El algoritmo genera una trayectoria en zigzag que recorre el área del polígono de manera sistemática, asegurando que cada punto dentro del polígono sea cubierto por la impresora. Sin embargo, no cubre áreas fuera del polígono, ya que los puntos que no cumplen con punto_en_poligono son descartados.

¿Cómo se ve afectada la impresión si se cambia la resolución?

La resolución es un compromiso entre calidad y tiempo de impresión. Una resolución demasiado baja puede dejar huecos en la impresión, mientras que una resolución demasiado alta puede hacer que el proceso sea innecesariamente lento.

- **Resolución alta (valores pequeños de res_x y res_y):**
  - **Ventaja:** La impresión es más precisa, ya que los puntos están más cerca entre sí, lo que permite cubrir mejor los detalles del modelo.
  - **Desventaja:** El tiempo de impresión aumenta significativamente, ya que se generan más puntos en la trayectoria. Además, el procesamiento puede volverse más lento debido al mayor número de cálculos.
- **Resolución baja (valores grandes de res_x y res_y):**

- **Ventaja**: La impresión es más rápida, ya que se generan menos puntos en la trayectoria.
- **Desventaja**: La calidad de la impresión disminuye, ya que los puntos están más separados, lo que puede dejar áreas sin cubrir o resultar en una impresión menos detallada.

¿En qué casos la estrategia diseñada no tiene un buen rendimiento?

La estrategia diseñada puede tener un rendimiento subóptimo en los siguientes casos:

- **Formas muy irregulares o complejas:**
  Para polígonos con bordes muy irregulares o formas extremadamente complejas, el algoritmo de zigzag puede generar trayectorias ineficientes, dejando áreas sin cubrir o requiriendo un tiempo de impresión excesivo.
- **Resolución inadecuada:**
  Si la resolución es demasiado baja, el algoritmo puede dejar huecos en la impresión, especialmente en áreas con detalles finos. Por otro lado, una resolución demasiado alta puede hacer que el proceso sea innecesariamente lento.
- **Procesamiento de archivos SVG mal formados:**
  Si el archivo SVG no está correctamente formado (por ejemplo, si los polígonos no están cerrados o si los puntos no están en el formato esperado), el algoritmo puede fallar o generar una trayectoria incorrecta.
- **Eficiencia computacional:**
  Para polígonos con un número muy grande de vértices o una resolución extremadamente alta, el algoritmo puede volverse lento debido a la gran cantidad de cálculos necesarios para verificar si cada punto está dentro del polígono.