INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

BIOINFORMATICS

PRÁCTICA 10.

MORPHOGENESIS

NOMBRE DEL ALUMNO: GARCÍA QUIROZ GUSTAVO IVAN

GRUPO: 7CV3

NOMBRE DEL PROFESOR: ROSAS TRIGUEROS JORGE LUIS

FECHA DE REALIZACIÓN DE LA PRÁCTICA: 08/04/2025

FECHA DE ENTREGA DEL REPORTE: 10/05/2025

Índice

1 Marco teórico.

1.1 Morfogénesis y Patrones Fractales en la Naturaleza

La morfogénesis, del griego "morphê" (forma) y "genesis" (creación), se refiere al proceso biológico que causa que un organismo desarrolle su forma. Este fenómeno fundamental en la biología del desarrollo explica cómo las células se organizan en patrones específicos para formar tejidos y órganos complejos.

1.1.1 Patrones y Autosimilitud en la Naturaleza

En la naturaleza, los patrones morfogenéticos frecuentemente exhiben propiedades de autosimilitud y complejidad a diferentes escalas, características definitorias de los fractales. Ejemplos de esto incluyen:

- La ramificación de los árboles y sus sistemas de raíces
- La estructura de los bronquiolos pulmonares
- Los patrones vasculares en hojas y sistemas circulatorios
- La forma de las costas y los ríos
- La estructura de los copos de nieve
- La formación de cristales y minerales

Estos patrones no son coincidencias, sino el resultado de procesos físicos y biológicos fundamentales que siguen reglas simples pero crean resultados sorprendentemente complejos.

1.1.2 Fractales

Un fractal es un objeto geométrico cuya estructura básica, fragmentada o aparentemente irregular, se repite a diferentes escalas. El término fue acuñado por el matemático Benoît Mandelbrot en 1975, aunque el estudio de estas formas comenzó mucho antes.

Las principales características de los fractales son:

1. Autosimilitud: Partes del objeto son similares al objeto completo.
2. Complejidad infinita: Al ampliar un fractal, aparecen nuevos detalles similares.
3. Dimensión fractal: Poseen dimensiones que no son números enteros.
4. Generación iterativa: Se generan mediante la aplicación repetida de reglas simples.

1.1.3 Fractales en la Biología y Bioinformática

La bioinformática utiliza los fractales como herramientas para:

1. **Modelado de estructuras biológicas**: Los L-sistemas (sistemas de Lindenmayer) permiten simular el crecimiento de plantas y otros organismos mediante gramáticas recursivas.
2. **Análisis de secuencias genómicas**: El análisis fractal de secuencias de ADN ha revelado patrones de autosimilitud que pueden relacionarse con la función genética.
3. **Estudio de morfogénesis**: Alan Turing propuso que patrones biológicos como las manchas en animales podrían surgir de sistemas de reacción-difusión, que matemáticamente producen estructuras similares a fractales.
4. **Modelado de ecosistemas**: Los patrones de distribución de especies y recursos naturales frecuentemente siguen distribuciones fractales.

1.1.4 La Ecuación de la Naturaleza

La "ecuación de la naturaleza" es un concepto que busca capturar matemáticamente cómo formas complejas en la naturaleza surgen de reglas simples aplicadas recursivamente. Esta aproximación ha sido explorada por científicos como Fereydoon Family, Stephen Wolfram y Fernando Galindo Soria.

La representación computacional de la ecuación de la naturaleza se puede expresar como:

```
ec(p, r, s, n) = {
    si n ≤ 0 entonces return
    aplicar r sobre p
    para cada elemento e resultante:
        ec(e, r, s, n-1)
}
```

Figura 1 Ecuación de la naturaleza

Donde:

- p representa el patrón o estructura inicial
- r representa las reglas de transformación
- s representa las condiciones de parada
- n representa el número de iteraciones

2 Material y equipo.

Para esta práctica se usaron las siguientes herramientas de software y hardware necesarias para realizar la práctica.

2.1.1 Hardware

- Computadora.

2.1.2 Software

- Navegador web
- Google colaboratory

3 Desarrollo de la práctica.

3.1 Planificación y Diseño

La práctica consistió en desarrollar un paisaje invernal nocturno utilizando algoritmos fractales para simular estructuras naturales. El desarrollo se realizó en Google Colab, utilizando como base el código proporcionado en el enlace de referencia (http://tinyurl.com/4erkxsu7), que implementa la "ecuación de la naturaleza".

El proceso de desarrollo siguió estas etapas:

1. Análisis del código base y comprensión del algoritmo fractal principal
2. Diseño conceptual del paisaje invernal nocturno
3. Implementación de los diferentes elementos fractales
4. Adaptación de los colores y efectos para un ambiente nocturno

3.2 Análisis del Código Base

El código original proporcionado implementaba una función recursiva ec() que genera estructuras fractales mediante la aplicación iterativa de transformaciones geométricas. Esta función tiene la siguiente estructura:

```
def ec(img,x,y,l,a,sl,da,n):
    if n<=0:
        return


    x2=x+l
    y2=y
    ar=math.radians(a)

    coseno=math.cos(ar)
    seno=math.sin(ar)
    xrot= (x2-x)*seno + (y2-y)*coseno
    yrot= (x2-x)*coseno - (y2-y)*seno

    x2=xrot + x
    y2=yrot + y

    cv2.line(img, (int(x),int(y)), (int(x2),int(y2)), (255,255,0), (1))
```

Este algoritmo permite generar diferentes patrones descomentando diferentes combinaciones de las llamadas recursivas y modificando los parámetros.

3.3 Diseño de los Elementos del Paisaje

Para crear nuestro paisaje invernal nocturno, se diseñaron los siguientes elementos fractales:

3.3.1 Diseño del Fondo

Se implementó un fondo gradiente para simular un cielo nocturno, utilizando una transición de azul oscuro a negro:

```
# Create night sky gradient from very dark blue to slightly lighter dark blue
for y in range(400):
    blue_val = min(50, 10 + y//10)
    img[y, :] = (blue_val, 20, 40)  # Dark night blue sky

# Create blue snow ground
img[400:, :] = (130, 110, 70)  # Blue-tinted snow for night
```

Figura 2 Diseño del Fondo

3.3.2 Generación de Estrellas y Luna

Las estrellas se generaron con un algoritmo que crea puntos brillantes de diferentes tamaños y añade efectos de brillo para algunas estrellas:

```
# Draw stars in the night sky
for _ in range(200):
    x = random.randint(0, 800)
    y = random.randint(0, 350)
    size = random.randint(1, 3)
    brightness = random.randint(150, 255)
    cv2.circle(img, (x, y), size, (brightness, brightness, brightness), -1)

    # Add twinkle effect to some stars
    if random.random() < 0.2:
        cv2.line(img, (x-size-1, y), (x+size+1, y), (brightness, brightness, brightness), 1)
        cv2.line(img, (x, y-size-1), (x, y+size+1), (brightness, brightness, brightness), 1)
```

Figura 3 Generación de Estrellas y Luna

La luna se implementó mediante una serie de círculos superpuestos y un efecto de halo:

```python
# Add moon
moon_x, moon_y = 100, 100
moon_radius = 40
# Full moon
cv2.circle(img, (moon_x, moon_y), moon_radius, (220, 220, 200), -1)
# Add some moon craters
for _ in range(5):
    crater_x = moon_x + random.randint(-moon_radius//2, moon_radius//2)
    crater_y = moon_y + random.randint(-moon_radius//2, moon_radius//2)
    if math.sqrt((crater_x-moon_x)**2 + (crater_y-moon_y)**2) < moon_radius*0.8:
        cv2.circle(img, (crater_x, crater_y), random.randint(3, 8), (200, 200, 180), -1)

# Add moonlight effect - subtle glow around the moon
for r in range(5, 30, 5):
    alpha = max(5, 50 - r*3)  # Decreasing opacity
    overlay = img.copy()
    cv2.circle(overlay, (moon_x, moon_y), moon_radius + r, (160, 160, 150), 5)
    cv2.addWeighted(overlay, alpha/100, img, 1 - alpha/100, 0, img)
```

Figura 4 implementación de la luna.

3.3.3 Desarrollo de la Función para Generar Copos de Nieve

Se desarrolló una función recursiva para generar copos de nieve con estructura fractal:

```python
# Function to draw a snowflake
def snowflake(img, x, y, size):
    angles = [0, 60, 120, 180, 240, 300]
    for angle in angles:
        rad = math.radians(angle)
        x2 = x + size * math.cos(rad)
        y2 = y + size * math.sin(rad)
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), (255, 255, 255), 1)

        # Draw branches
        branch_size = size * 0.4
        branch_angle1 = rad + math.radians(30)
        branch_angle2 = rad - math.radians(30)

        xb1 = x2 - branch_size * math.cos(branch_angle1)
        yb1 = y2 - branch_size * math.sin(branch_angle1)

        xb2 = x2 - branch_size * math.cos(branch_angle2)
        yb2 = y2 - branch_size * math.sin(branch_angle2)

        cv2.line(img, (int(x2), int(y2)), (int(xb1), int(yb1)), (255, 255, 255), 1)
        cv2.line(img, (int(x2), int(y2)), (int(xb2), int(yb2)), (255, 255, 255), 1)
```

Figura 5 Función para Generar Copos de Nieve

3.3.4 Implementación de la Función para Árboles Fractales

Para los árboles fractales, se modificó la función original para crear estructuras ramificadas que simulan árboles con nieve:

```python
# Function to draw a fractal tree (modified for night scene)
def fractal_tree(img, x, y, l, a, sl, da, n, color=(60, 40, 20)):  # Darker tree color for night
    if n <= 0:
        return

    x2 = x + l * math.cos(math.radians(a))
    y2 = y - l * math.sin(math.radians(a))  # Using - because y increases downward

    # Draw the branch
    thickness = max(1, int(n/2))
    cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), color, thickness)

    # If it's one of the last branches, add bluish snow on top for night scene
    if n <= 2:
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), (180, 180, 210), thickness)

    # Recursive calls for branches
    fractal_tree(img, x2, y2, l*sl, a+da, sl, da, n-1, color)
    fractal_tree(img, x2, y2, l*sl, a-da, sl, da, n-1, color)

    # For more complex trees, add a third branch occasionally
    if n > 3 and random.random() < 0.3:
        fractal_tree(img, x2, y2, l*sl*0.8, a, sl, da, n-2, color)
```

Figura 6 Función para Árboles Fractales

3.3.5 Desarrollo de la Función para Montañas Fractales

Las montañas se implementaron utilizando un algoritmo de subdivisión recursiva que sigue el concepto de "desplazamiento del punto medio":

```python
def mountain(img, base_x, base_width, height, roughness=0.6):
    points = [(base_x, 400)]

    # Generate a fractal mountain line
    def divide_line(x1, y1, x2, y2, depth):
        if depth <= 0:
            return

        mid_x = (x1 + x2) / 2
        mid_y = (y1 + y2) / 2 - random.uniform(0, 1) * roughness * abs(x2 - x1)
        mid_y = max(mid_y, 200)  # Don't go too high

        points.append((mid_x, mid_y))

        divide_line(x1, y1, mid_x, mid_y, depth-1)
        divide_line(mid_x, mid_y, x2, y2, depth-1)

    # End point of the mountain range
    points.append((base_x + base_width, 400))

    # Divide the mountain line to create a fractal-like shape
    divide_line(base_x, 400, base_x + base_width, 400, 6)

    # Sort points by x coordinate
    points.sort()

    # Convert to numpy array for fillPoly
    pts = np.array([points], dtype=np.int32)

    # Fill the mountain with a gradient
    cv2.fillPoly(img, pts, (160, 160, 200))  # Light grayish-blue for distant mountains
```

```python
    # Add snow caps
    snow_pts = []
    for x, y in points:
        if y < 350:  # Only add snow to higher parts
            snow_pts.append([x, y])

    if snow_pts:
        snow_line = []
        for i, (x, y) in enumerate(snow_pts):
            snow_line.append([x, y])
            # Add a point below for the snow cap thickness
            if i < len(snow_pts) - 1:
                snow_line.append([x, y + random.randint(5, 15)])

        snow_line_np = np.array([snow_line], dtype=np.int32)
        cv2.fillPoly(img, snow_line_np, (255, 255, 255))
```

Figura 7 Función para Montañas Fractales

3.3.6 Implementación de la Cabaña y Efectos de Luz

Para añadir un elemento focal al paisaje, se implementó una cabaña con efectos de luz que emanan de su ventana:

```
# Draw a small cabin in the middle
cabin_x, cabin_y = 400, 480
cabin_width, cabin_height = 120, 80

# Cabin base - darker for night
cv2.rectangle(img, (cabin_x, cabin_y), (cabin_x + cabin_width, cabin_y + cabin_height), (40, 20, 10), -1)

# Cabin roof (triangle) - darker for night
roof_points = np.array([
    [cabin_x - 10, cabin_y],
    [cabin_x + cabin_width + 10, cabin_y],
    [cabin_x + cabin_width//2, cabin_y - cabin_height//1.5]
], dtype=np.int32)
cv2.fillPoly(img, [roof_points], (50, 25, 10))

# Add snow on the roof - blue-tinted for night
snow_points = np.array([
    [cabin_x - 10, cabin_y],
    [cabin_x + cabin_width + 10, cabin_y],
    [cabin_x + cabin_width//2, cabin_y - cabin_height//1.5],
    [cabin_x + cabin_width//2, cabin_y - cabin_height//1.5 + 5],
    [cabin_x + cabin_width + 5, cabin_y - 5],
    [cabin_x - 5, cabin_y - 5]
], dtype=np.int32)
cv2.fillPoly(img, [snow_points], (160, 160, 200))
```

Figura 8 Cabaña y Efectos de Luz

4 Resultados

Una vez implementados todos los elementos, se procedió a integrarlos en la escena final. Los elementos se posicionaron siguiendo principios de composición visual y considerando la perspectiva:

- Las montañas en el fondo, con diferentes tamaños y posiciones para dar sensación de profundidad.
- Los árboles distribuidos en grupos, con variaciones aleatorias en altura y estructura.
- La cabaña como punto focal central con luz que contrasta con la oscuridad.
- Copos de nieve y estrellas dispersos para dar vida a la escena.
- El estanque posicionado para reflejar la luz de la luna y crear un punto de interés adicional.

El resultado final es un paisaje nocturno invernal que demuestra cómo los algoritmos fractales pueden simular efectivamente elementos naturales.

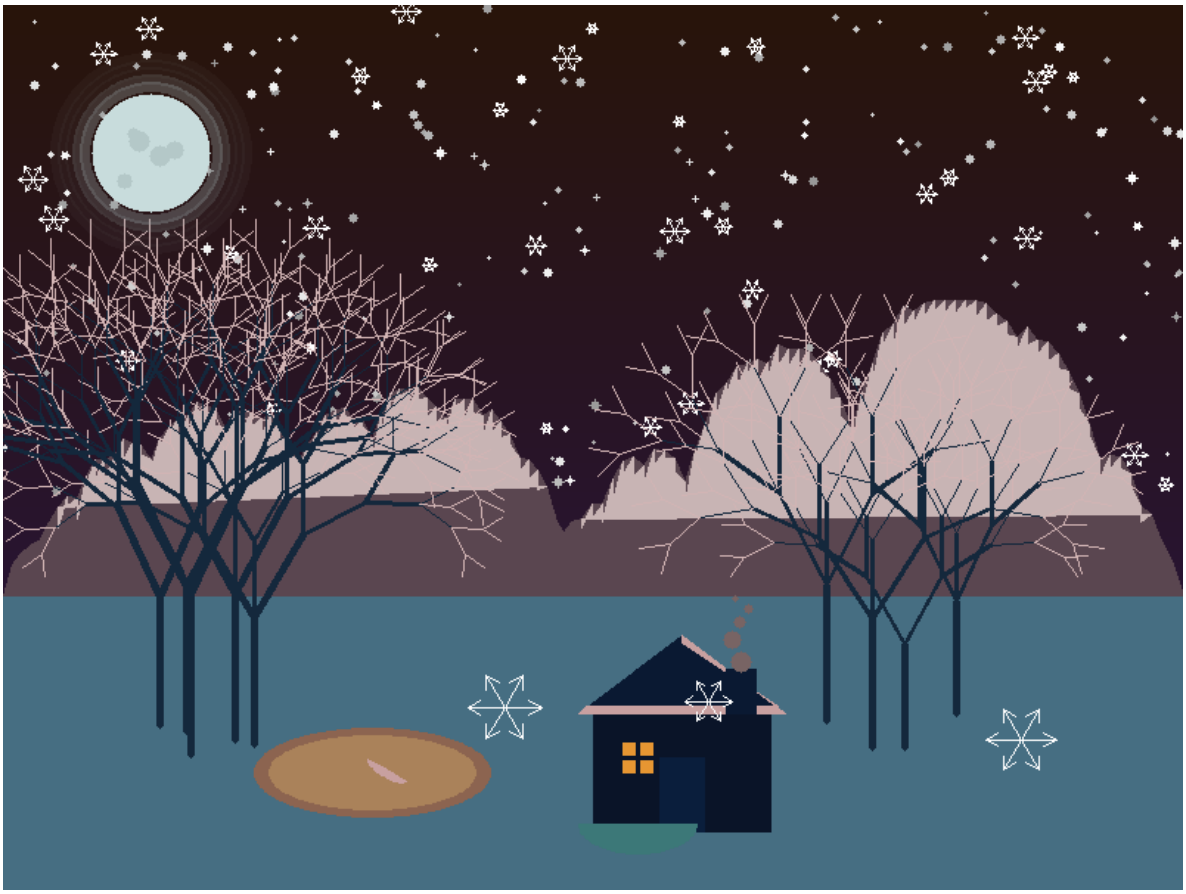

Figura 9 Resultados

4.1 Desafíos y soluciones

Durante el desarrollo de la práctica, se enfrentaron varios desafíos técnicos:

1. Ajuste de parámetros: Encontrar los valores óptimos para los parámetros de las funciones fractales requirió experimentación para lograr estructuras naturales convincentes.
2. Efectos de iluminación: Simular la iluminación nocturna y los efectos de luz presentó desafíos para mantener el balance entre visibilidad y realismo.
3. Rendimiento: Los algoritmos fractales son intensivos computacionalmente, por lo que fue necesario limitar la profundidad de la recursión en algunas funciones.
4. Integración visual: Conseguir que todos los elementos fractales se integraran coherentemente en una escena unificada requirió ajustes cuidadosos en colores, escalas y posiciones.

Estos desafíos se resolvieron mediante prueba y error de diseño.

5 Conclusiones y recomendaciones.

La realización de esta práctica sobre morfogénesis y patrones fractales ha permitido obtener conclusiones. La generación de un paisaje de invierno nocturno utilizando algoritmos fractales demuestra cómo estructuras complejas y visualmente realistas pueden emerger de reglas simples aplicadas recursivamente. Este principio es análogo a los procesos morfogenéticos en sistemas biológicos, donde instrucciones genéticas relativamente simples generan estructuras orgánicas complejas.

El uso de Python y bibliotecas como OpenCV demuestra cómo la programación puede servir como herramienta para modelar y simular procesos biológicos fundamentales, permitiendo experimentar con parámetros que serían difíciles de manipular en sistemas reales.

A partir de la experiencia adquirida en esta práctica, se proponen las siguientes recomendaciones para futuros trabajos. Se recomienda experimentar sistemáticamente con los parámetros de los algoritmos fractales para comprender mejor su impacto en las estructuras resultantes. Esto podría incluir la creación de interfaces interactivas que permitan la modificación en tiempo real de variables como ángulos de ramificación, factores de escala y niveles de recursión. Desarrollar métodos para calibrar y validar los modelos fractales con datos empíricos de estructuras biológicas reales, como imágenes microscópicas de tejidos o secuencias genómicas, aumentaría significativamente su relevancia biológica.

6 Referencias

- Barnsley, M. F. (2014). *Fractals Everywhere*. Academic Press.
- Falconer, K. (2004). *Fractal Geometry: Mathematical Foundations and Applications*. John Wiley & Sons.
- Galindo Soria, F. (2023). *La Ecuación de la Naturaleza*. Recuperado de https://www.fgalindosoria.com/ecuaciondelanaturaleza/
- Goodwin, B. C. (2001). *How the Leopard Changed Its Spots: The Evolution of Complexity*. Princeton University Press.
- Kauffman, S. A. (1993). *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press.