INSTITUTO TECNOLÓGICO SUPERIOR PROGRESO

Carrera:

Ingeniería en sistemas computacionales

Materia:

Graficación

Actividad:

Reporte de practica 2-2

Docente:

Dr. Holzen Atocha Martínez García

Alumno:

David Ezequiel Caballero González

Fecha de entrega:

Lunes 8 de septiembre de 2025

Práctica 2-2

Introducción

En esta práctica se desarrolló un programa en Processing que dibuja polígonos regulares utilizando vértices conectados mediante las funciones ***beginShape()*** y ***endShape(CLOSE)***. A diferencia de la práctica 1-1, donde se trazaban líneas entre puntos con ***line()***, este enfoque permite construir figuras de forma más limpia y modular. El objetivo es reforzar el uso de coordenadas polares y funciones trigonométricas (***cos() y sin()***) para posicionar los vértices de manera equidistante alrededor de una circunferencia imaginaria. Esta técnica es fundamental en la graficación computacional, ya que permite representar figuras geométricas con precisión matemática y visual.

- El código puede verse en el repositorio en GitHub: Graficación_Práctica 2_Práctica 2-2

Objetivos

Desarrollar un programa en Processing que permita dibujar polígonos de ***n*** lados con ayuda de las funciones ***beginShape()*** y ***endShape(CLOSE)***.

Marco teórico

Un polígono regular es una figura geométrica cuyos lados y ángulos son iguales. Para construirlo en un entorno gráfico como Processing, se distribuyen los vértices de forma equidistante alrededor de un círculo y esto se logra dividiendo el círculo completo (***2π radianes***) entre el número de lados del polígono, obteniendo así el ángulo entre vértices.
Las funciones trigonométricas ***cos()*** y ***sin()*** permiten calcular las coordenadas (***x, y***) de cada vértice con respecto al centro de la figura, permitiendo dibujar líneas entre cada par de puntos sin la necesidad de la función ***line()*** como en la práctica 1-1, esta vez se utiliza ***beginShape()*** para iniciar la figura y ***vertex()*** para agregar cada punto. Finalmente, ***endShape(CLOSE)*** cierra automáticamente el polígono, conectando el último vértice con el primero, este método mejora la legibilidad del código y facilita la creación de figuras más complejas.

Procedimiento

1. Retomando el código de la practica 1-1, se deja tal cual la función ***void setup***. Tambien se debe de cambiar el nombre de la función a ***poligonShape()***.

```
void setup() {
  size(500, 500); // Tamaño de la ventana
  background(255); // Fondo blanco
  stroke(0); // Color del trazo: negro
  noFill(); // Sin relleno en el polígono

  // Ejemplo: polígono centrado en (250, 250), radio 100, con 5 lados
  poligonShape(250, 250, 100, 5);
}
```

2. Ahora, a la función ***void poligon*** la llamamos ***poligonShape()*** que recibirá los mismos 4 parámetros puestos anteriormente. También se debe dejar tal cual la variable ***anguloV***.

```
void poligonShape(int cx, int cy, int r, int n) {
  float anguloV = TWO_PI / n; // Ángulo entre vértices
```

Por último, en vez de usar la función ***line()***, usamos la función ***beginShape()*** y ***endShape(CLOSE)*** para unir los vértices de la variable ***anguloV***, usando un bucle ***for*** que se repite ***n*** veces para calcular las coordenadas de cada vértice.
En cada iteración, se calcula el ángulo actual multiplicando ***i * anguloV***, luego se obtiene la posición (***x, y***) del vértice usando funciones trigonométricas: ***x = cx + r * cos(angulo) y = cy + r * sin(angulo)*** y se lanza el vértice con ***vertex(x, y)***.
Por último, se termina la figura con ***endShape(CLOSE)***, lo que conecta automáticamente el último vértice con el primero y cierra el polígono.

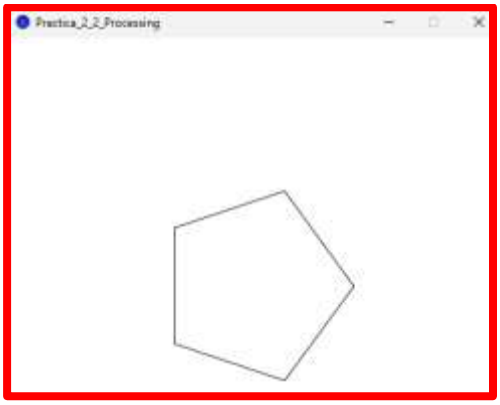
```
void poligonShape(int cx, int cy, int r, int n) {
  float anguloV = TWO_PI / n; // Ángulo entre vértices

  beginShape(); // Inicia forma libre

  for (int i = 0; i < n; i++) {
    float angulo = i * anguloV;
    float x = cx + r * cos(angulo);
    float y = cy + r * sin(angulo);
    vertex(x, y); // Añade vértice
  }

  endShape(CLOSE); // Cierra la figura
}
```

Materiales y equipos utilizados

- Laptop
- Processing
- Página oficial de Processing

Resultados

Se pudo apreciar correctamente el polígono, de igual forma que en la practica 1.



Análisis

Esta práctica permitió visualizar cómo las funciones trigonométricas se aplican directamente en la programación gráfica. El ángulo entre vértices es clave para mantener la regularidad de la figura y el uso de coordenadas polares simplifica la ubicación de los puntos y con la implementación de ***beginShape()*** y ***vertex()*** no solo facilita el trazado de polígonos, sino que también abre la puerta a la creación de formas irregulares, estrellas o superficies en 3D. Se noto también que el código es más compacto y reutilizable, lo cual es ideal para proyectos más avanzados.

Discusión

Comparando esta práctica con la 1-1, el uso de ***beginShape()*** representa una mejora significativa en la estructuración del código, mientras que antes se dibujaban líneas manualmente, ahora los vértices se conectan automáticamente, haciendo el código más legible y eficiente. El principio matemático sigue siendo el mismo: dividir un círculo en ángulos iguales, demostrando que comprender los fundamentos trigonométricos permite resolver problemas gráficos con distintos enfoques y que Processing ofrece herramientas poderosas para representar ideas matemáticas de forma visual.

Conclusión

La práctica 2-2 fue útil para profundizar en el uso de Processing en la generación de polígonos regulares. El programa demostró que el trazado de figuras geométricas puede lograrse de manera más eficiente utilizando ***beginShape()*** y ***vertex()*** en lugar de ***line()***. Se reforzó la conexión entre trigonometría y graficación computacional, mostrando cómo los cálculos matemáticos se transforman en representaciones visuales claras y precisas. Además, se destacó la importancia de escribir código modular y reutilizable para proyectos de mayor complejidad.

Referencias

- Processing Foundation. (n.d.). beginShape(). Processing.org. https://processing.org/reference/beginShape_.html