INSTITUTO TECNOLÓGICO SUPERIOR PROGRESO

Carrera:

Ingeniería en sistemas computacionales

Materia:

Graficación

Actividad:

Reporte de practica 1-1

Docente:

Dr. Holzen Atocha Martínez García

Alumno:

David Ezequiel Caballero González

Fecha de entrega:

Lunes 8 de septiembre de 2025

Práctica 1-1

Introducción

En esta práctica se desarrolló un programa que dibuja polígonos de ***n*** lados usando funciones básicas como ***setup()*** y una función personalizada llamada *poligon()*. Los vértices de la figura se calculan con ***cos() y sin()***, que permiten ubicarlos alrededor de una circunferencia imaginaria para asegurar que todos los lados sean iguales y estén distribuidos uniformemente y para unir esos puntos y formar el polígono se utiliza la función ***line()***, que dibuja cada segmento entre vértices.

La práctica sirve para ver cómo se relacionan las matemáticas con la programación y también para aprender la importancia de escribir un código ordenado, modular y fácil de reutilizar. Además, permite experimentar cómo cambia la figura al variar el número de lados y visualizar de manera sencilla la conexión entre conceptos matemáticos y su representación digital.

- El código puede verse en el repositorio en GitHub: Graficación Práctica 1 Práctica 1-1

Objetivos

Desarrollar un programa en Processing que permita dibujar polígonos de ***n*** lados.

Marco teórico

Processing es un lenguaje de programación basado de Java y está pensado para crear dibujos, animaciones y cosas interactivas. Su editor (PDE) es muy práctico porque te deja escribir código de manera rápida y sin tantas complicaciones, perfecto si apenas estás empezando con la programación visual.

En esta práctica usamos la función ***setup()*** para preparar el espacio donde vamos a dibujar y una función propia llamada ***poligon()***, que se encarga de crear polígonos regulares. Para trazar las líneas se utiliza ***line(x1, y1, x2, y2)***, que une cada punto del polígono y las posiciones de esos puntos se calculan con las funciones trigonométricas ***cos()*** y ***sin()***, que básicamente sirven para colocar cada vértice alrededor de una circunferencia imaginaria, usando radianes en lugar de grados. La constante ***TWO_PI*** representa un giro completo (***360° o 2π radianes***), qué si la divides entre el número de lados del polígono, obtienes el ángulo entre un vértice y el siguiente. Este truco se usa mucho en gráficos por computadora para generar figuras geométricas exactas.

Procedimiento

1. Implementar la función ***void setup*** en donde indicaremos donde iniciara el programa, definiendo el tamaño de la ventana, el color del fondo, el color del trazo y que no se rellene el polígono.

```
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
```

2. Ahí mismo en el ***void setup*** definimos la función ***poligon*** que crearemos más adelante. Podemos poner los parámetros que queramos de esta manera: centro (dos enteros), radio (entero) y número de lados del polígono (entero).

```
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

  // Polígono centrado en (250, 250), radio 100, con 6 lados
  poligon(250, 250, 100, 6);
}
```

3. Ahora se crea la función ***void poligon*** que recibe los 4 parámetros puestos anteriormente (250,250,100,6), para eso los definimos como enteros con las variables ***cx***, ***cy***(coordenadas del centro del polígono), ***r***(radio) y ***n***(número de lados del polígono).

```
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)
  poligon(250, 250, 100, 6);
}

void poligon(int cx, int cy, int r, int n) {

}
```

4. Luego, para poder crear un polígono primero necesitamos saber el ángulo entre cada vértice para saber dónde trazar la línea entre los vértices, para ello se utiliza la constante predefinida ***TWO_PI*** de Processing que representa el valor de ***2 × π (pi)***, es decir, un círculo completo en radianes, entonces a esta constante la dividimos entre los números de lados del polígono (***n***). Este procedimiento lo guardamos en una variable con el nombre ***anguloV***(ángulo del vértice) con el tipo de dato ***float.***

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

5. Hasta ahora se tiene el ángulo entre los vértices, pero falta la ubicación de esos vértices o puntos para luego unirlos, para ello sabemos que un polígono regular tiene todos sus vértices equidistantes del centro y separados por el mismo ángulo, entonces con

trigonometría se pueden calcular las posiciones de estos vértices para luego guardar la coordenada en una variable(***x1,y1***).

```
void poligon(int cx, int cy, int r, int n) {
  float anguloV = TWO_PI / n; // Ángulo entre vértices
  float x1 = cx + r * cos(0);
  float y1 = cy + r * sin(0);
```

6. Con el paso anterior se tiene la coordenada de un vértice del polígono, ahora solo se tiene que encontrar la ubicación del siguiente vértice para unirlos, para ello utilizaremos la lógica anterior con la diferencia que se utilizara un bucle ***for*** que se repetirá ***n*** veces, sabiendo que ***n*** es el numero de lados y también se utilizara la variable ***i*** que representa el índice del vértice actual(empieza en 1 porque el vértice anterior (i = 0) ya se calculó antes del bucle). Sabiendo eso, en cada vuelta se calcula la posición del siguiente vértice (x2, y2) y se dibuja una línea desde el anterior con la función ***line(x1, y1, x2, y2).***

```
void poligon(int cx, int cy, int r, int n) {
  float anguloV = TWO_PI / n; // Ángulo entre vértices
  float x1 = cx + r * cos(0);
  float y1 = cy + r * sin(0);

  for (int i = 1; i <= n; i++) {
    float angulo = i * anguloV;
    float x2 = cx + r * cos(angulo);
    float y2 = cy + r * sin(angulo);

    line(x1, y1, x2, y2); // Traza línea entre vértices
    x1 = x2;
    y1 = y2;
  }
}
```

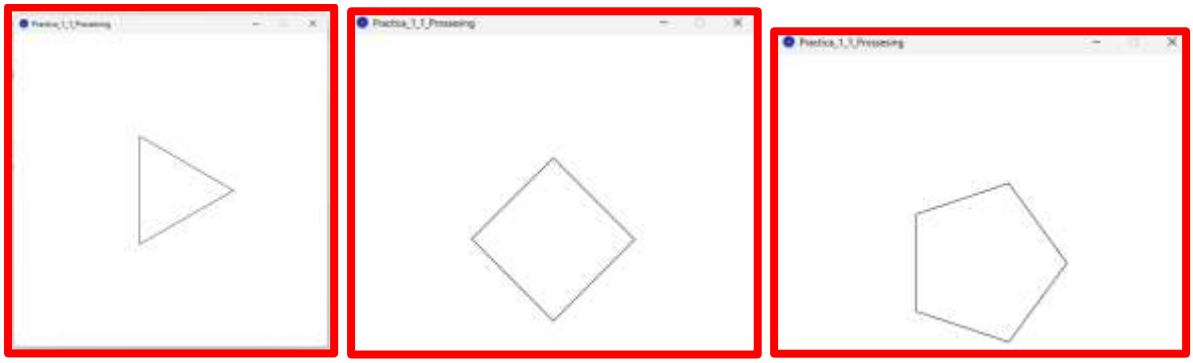
Nota: después de dibujar la línea se deben de actualizar las variables ***x1*** y ***x2*** para empezar desde el siguiente punto.

Materiales y equipos utilizados

- Laptop
- Processing
- Video de apoyo en YouTube
- Página oficial de Processing

Resultados

Se pudo apreciar correctamente el polígono correspondiendo al numero de lados que pongamos (mínimo 3 lados para visualizar un polígono).



Análisis

En esta práctica se logró dibujar un polígono regular con Processing. El programa crea una ventana de 500x500 píxeles con fondo blanco y líneas negras, y en el centro (250, 250) dibuja un polígono de 5 lados con un radio de 100.La función ***poligon()*** se encarga de calcular los puntos del polígono usando ***cos()*** y ***sin()***, que sirven para ubicar cada vértice como si estuvieran sobre una circunferencia imaginaria y después, se van uniendo esos puntos con líneas para darle forma a la figura.

Al correr el código, el polígono aparece bien formado, con lados iguales y simétricos , además es fácil ver cómo el número de lados (***n***) cambia la figura: con 3 lados se dibuja un triángulo, con 6 un hexágono, y así sucesivamente.

El código es sencillo de modificar para probar distintas formas y ayuda a entender la relación entre ángulos, radio y centro al momento de construir un polígono. Esta práctica muestra cómo las matemáticas se aplican de forma directa en los gráficos y lo mucho que Processing facilita ese trabajo.

Discusión

En esta práctica me di cuenta de cómo las funciones trigonométricas ayudan a construir figuras geométricas con mucha precisión en Processing; El polígono salió bien, con todos sus lados iguales y bien centrado, esto muestra que calcular el ángulo entre los vértices y usar ***cos()*** y ***sin()*** es una forma muy práctica de ubicar puntos alrededor de un círculo.

Al cambiar el número de lados también fue fácil notar la diferencia: con 3 lados aparece un triángulo, con 4 un cuadrado, con 6 un hexágono, y así se pueden generar distintas figuras, esto deja claro que el código es flexible y se adapta rápido a lo que uno quiera dibujar y además, la práctica ayudó a entender mejor cómo se organiza un programa en Processing, cómo usar funciones para no repetir código y cómo una misma función puede servir para varios casos.

Aunque el ejercicio fue sencillo, permitió aplicar conceptos de matemáticas de manera visual, lo cual hace mucho más fácil aprenderlos.

Conclusión

La práctica resultó bastante útil porque permitió ver cómo la trigonometría se puede aplicar directamente en la programación gráfica y se logró crear una función que dibuja polígonos de cualquier número de lados usando coordenadas polares y funciones como ***cos()*** y ***sin()***.

También se reforzó el uso de Processing como una buena herramienta para representar ideas matemáticas con gráficos y quedó claro lo importante que es organizar el código de forma ordenada.

Referencias

- Processing Foundation. (s.f.). Regular Polygon. Processing.org. Recuperado de https://processing.org/examples/regularpolygon.html
- Air Room. (2019, febrero 6). PROCESSING: 8.8 Función Polígono [En español] [Video]. YouTube. Recuperado de https://youtu.be/_dH19YsxVNg
- del Valle Hernández, L. (s.f.). *Processing, el lenguaje para gráficos*. Programar Fácil Podcast #80. Recuperado de https://programarfacil.com/podcast/80-processing-lenguaje-para-graficos/