INSTITUTO TECNOLÓGICO SUPERIOR PROGRESO

Carrera:

Ingeniería en sistemas computacionales

Materia:

Graficación

Actividad:

Ada 2.2 - Reporte de prácticas PROCESSING 3 y 4

Docente:

Dr. Holzen Atocha Martínez García

Alumno:

David Ezequiel Caballero González

Fecha de entrega:

Miércoles 17 de septiembre de 2025

Prácticas 3 y 4

Introducción

En este reporte se muestra el desarrollo y los resultados de las prácticas 3 y 4 de la materia de Graficación. Ambas prácticas se enfocaron en la manipulación y visualización de gráficos utilizando el lenguaje de programación Processing, abordando dos áreas clave: el procesamiento de imágenes y las transformaciones 2D. La práctica 3 se centró en la carga y el análisis de imágenes, mientras que la práctica 4 consistió en la modificación de códigos previos para incorporar transformaciones 2D.

- Los códigos pueden verse en el repositorio en GitHub: Graficación Prácticas 3 y 4

Objetivos

- Cargar una imagen en el entorno de Processing y mostrarla en la ventana de visualización, asegurando que se adapte al tamaño del lienzo manteniendo sus proporciones originales y que se posicione de forma centrada.
- Desarrollar un programa que calcule el histograma de una imagen y lo visualice.
- Aplicar transformaciones 2D para modificar y mejorar códigos existentes.

Marco teórico

- ***Imágenes***
  En Processing, la clase "***PImage***" se utiliza para manejar objetos de tipo imagen. Las imágenes pueden ser cargadas desde archivos utilizando la función "***loadImage()***" y visualizadas con la función "***image()***". Para la manipulación de píxeles, se pueden usar los métodos "***get()***" para obtener el color de un píxel específico, y "***set()***" para cambiar su valor. La clase "***PImage***" también ofrece los atributos "***width***" y "***height***" para conocer las dimensiones de la imagen.

  El trabajo con colores se simplifica con el tipo de dato color y la función "***color(r, g, b)***", que permite crear un color a partir de sus valores RGB. Las funciones "***red()***", "***green()***" y "***blue()***" son útiles para extraer los valores de cada canal de un color dado.

- ***Histograma***
  Un histograma es una representación gráfica que muestra la distribución de los valores de intensidad de los píxeles en una imagen. Para calcularlo, se puede convertir la imagen a escala de grises usando la función "***filter(GRAY)***". De esta manera, cada píxel tiene un único valor de brillo, que puede ser accedido a través del canal rojo. Luego, se recorren los píxeles para contar la frecuencia de cada valor de intensidad en un arreglo.

- ***Transformaciones 2D***
  Las transformaciones geométricas 2D, como la traslación, la rotación y el escalado, son fundamentales para manipular modelos y sistemas de coordenadas. Processing gestiona estas transformaciones a través de una matriz interna llamada matriz de modelo/vista.

Inicialmente, esta matriz es la matriz identidad, lo que significa que no aplica ninguna transformación.
Las funciones "***translate(tx, ty)***", "***rotate(angulo)***", y "***scale(sx, sy)***" modifican esta matriz interna. Es importante destacar que estas funciones se aplican en un orden inverso al que se codifican, ya que cada llamada multiplica la matriz de transformación actual por la nueva matriz. Para gestionar múltiples transformaciones y volver a estados anteriores, se utilizan las funciones "***pushMatrix()***" y "***popMatrix()***", que guardan y restauran el estado de la matriz de transformación en una pila.

Procedimientos

Práctica 3-1

1. Primero se define las características de nuestro lienzo en el "***void setup***".

```
void setup(){
  background(0);    //Fondo negro
  size(250,400);    //Tamaño del lienzo
```

2. Luego se crea un objeto tipo imagen con "***PImage***", se le pone el nombre "imagen" y a ese objeto creados se le carga o una imagen con "***loadImage()***". La imagen a cargar debe estar en el directorio data donde radica el programa de Processing.

```
//Crea un objeto tipo imagen llamado "image"
//Y con "loadImage()" carga la imagen en el objeto "imagen"
PImage imagen = loadImage("C:Marca de agua_Maui y Shirley.jpeg");
```

3. Para terminar, se visualiza la imagen con "***image***". Se puede cambiar las dimensiones del lienzo y de la imagen a voluntad para apreciarla de diferentes tamaños.

```
  //Visualiza la imagen que tiene el objeto "imagen"
  image(imagen,0,50,250,300);
}
```

Práctica 3-2

1. Antes de iniciar el programa, se declaran tres variables globales:
   - ***"PImage img;"*** para almacenar la imagen que se va a utilizar.
   - ***"int[] histograma = new int[256];"*** para contar la frecuencia de cada nivel de brillo (0–255).
   - ***"int histogramaMax = 0;"*** para registrar el valor máximo del histograma, útil para escalar la visualización.

```
PImage img;
int[] histograma = new int[256];
int histogramaMax = 0;
```

2. En la función "***setup()***", se define el tamaño de la ventana con "***size(300,500)***". Luego se carga la imagen desde la ruta local con "***loadImage("C:Ejemplo.jpg")***" y se convierte a escala de grises usando "***filter(GRAY)***", simplificando el análisis de brillo, ya que cada píxel tendrá un único valor de intensidad. También se llama a la función "***calcularHistograma()***" que se explicara más adelante.

```
void setup() {
  size(300,500); // Tamaño total de la ventana
  img = loadImage("C:Ejemplo.jpg"); // Imagen
  img.filter(GRAY); // Convertir a escala de grises

  calcularHistograma(); //LLamado de la función que calcula el histograma
}
```

3. Se crea la función "***calcularHistograma()***", que recorre cada píxel de la imagen usando dos bucles ***for***. Se obtiene el valor de brillo con "***int(red(img.get(x, y)))***", ya que en escala de grises los tres canales son iguales. Cada valor se acumula en el arreglo histograma, y se actualiza "***histogramaMax***" si se encuentra una frecuencia mayor.

```
void calcularHistograma() {
  for (int x = 0; x < img.width; x++) {
    for (int y = 0; y < img.height; y++) {
      int brillo = int(red(img.get(x, y))); // Valor de gris
      histograma[brillo]++;
      if (histograma[brillo] > histogramaMax) {
        histogramaMax = histograma[brillo];
      }
    }
  }
}
```

4. En una función "***draw()***", se establece un fondo negro con "***background(0)***" y se muestra la imagen en la parte superior del lienzo con "***image(img, 0, 0, width, height - 110)***", dejando espacio en la parte inferior para el histograma. También se llama la función "***dibujarHistograma()***" que se explicara mas adelante.

```
void draw() {
  background(0); // Fondo negro

  // Mostrar imagen en la parte superior
  image(img, 0, 0, width, height - 110);

  // Dibujar histograma en la parte inferior
  dibujarHistograma();
}
```

5. Por último, se crea la función "***dibujarHistograma()***" que dibuja líneas verticales blancas para cada nivel de brillo. Se usa "***map()***" para escalar la posición horizontal "***(x)"*** y la altura "***(h)"*** de cada línea según el valor en "***histograma[i]***" y el máximo registrado en "**histogramaMax**". El histograma se dibuja en la parte inferior del lienzo, ocupando un área de 256 píxeles de ancho por 100 de alto.

```
void dibujarHistograma() {
  int ubicacionX = 0;
  int ubicacionY = height - 100;
  int ancho = 256;
  int alto = 100;

  stroke(255); // Color blanco
  for (int i = 0; i < histograma.length; i++) {
    float x = map(i, 0, 255, ubicacionX, ubicacionX + ancho);
    float h = map(histograma[i], 0, histogramaMax, 0, alto);
    line(x, ubicacionY + alto, x, ubicacionY + alto - h);
  }
}
```

Práctica 4

Modificación de la práctica 2-1

En el código original de la práctica 2-1, las funciones seno y coseno se graficaban usando la función "***map()***" para convertir manualmente el rango de valores matemáticos al tamaño de la ventana. En esta modificación se elimina ese mapeo y en su lugar se ajustó el sistema de coordenadas mediante transformaciones.

1. Primero se utiliza "***translate(0, h/2)***" para trasladar el origen al centro vertical de la ventana, logrando que el eje ***X*** quede alineado en la mitad de la pantalla y que las funciones trigonométricas se dibujen alrededor de dicho eje. Posteriormente se aplicó "***scale(w / (2PI), -h/2)***", lo que permitió que el intervalo de ángulos de ***0 a 2PI*** ocupe todo el ancho de la ventana y que los valores verticales de -1 a 1 se adapten al alto disponible. El signo negativo en el eje ***Y*** fue necesario para invertir la orientación, de modo que los valores positivos se representaran hacia arriba, tal como en el sistema cartesiano convencional.

```
  // Guardamos estado original de la matriz
  pushMatrix();

  // Cambiar el sistema de coordenadas:
  // Trasladamos el origen a (0, height/2) -> centro vertical
  // Escalamos en X para mapear [0, 2PI] a [0, width]
  // Escalamos en Y para mapear [-1, 1] a [height, 0] (invirtiendo eje Y)
  translate(0, h/2);
  scale(w / (2*PI), -h/2);

  // Dibujar seno en rojo
  stroke(255, 0, 0);
  strokeWeight(0.03);
  dibujarSeno();

  // Dibujar coseno en verde
  stroke(0, 255, 0);
  strokeWeight(0.03);
  dibujarCoseno();

  // Restaurar sistema de coordenadas original
  popMatrix();
}
```

2. Finalmente, se redefinieron las funciones "***dibujarSeno()***" y "***dibujarCoseno()***" para que trabajen directamente con los valores de seno y coseno, recorriendo los ángulos de ***0 a 2*PI*** en pasos pequeños. Con esto se consiguió que las curvas se generen correctamente bajo el nuevo sistema de coordenadas sin necesidad de conversiones adicionales.

```
void dibujarSeno() {
  float prevX = 0;
  float prevY = sin(0);
  for (float angulo = 0; angulo <= TWO_PI; angulo += 0.01) {
    float x = angulo;
    float y = sin(angulo);
    line(prevX, prevY, x, y);
    prevX = x;
    prevY = y;
  }
}

void dibujarCoseno() {
  float prevX = 0;
  float prevY = cos(0);
  for (float angulo = 0; angulo <= TWO_PI; angulo += 0.01) {
    float x = angulo;
    float y = cos(angulo);
    line(prevX, prevY, x, y);
    prevX = x;
    prevY = y;
  }
}
```

Modificación de la práctica 3-1

1. En el código original de la práctica 3-1, la imagen se dibujaba con dimensiones fijas, lo que podía provocar que no aprovechara todo el espacio de la ventana o que se deformara si las proporciones no coincidían. Para corregir esto, se calcularon dos factores de escala: uno en el eje X y otro en el eje ***Y***, resultantes de dividir el ancho y el alto de la ventana entre las dimensiones originales de la imagen.

```
// Calcular escalado proporcional
float escalaX = width / float(imagen.width);
float escalaY = height / float(imagen.height);
```

2. Se eligió el menor de ambos valores como factor de escala global, asegurando que la imagen conserve su proporción sin estiramientos. Luego, a partir de este escalado, se

calcularon nuevas dimensiones y las posiciones "***posX***" y "***posY***" necesarias para centrar la imagen en la ventana.

```
// Factor de escala global
float escala = min(escalaX, escalaY);

// Centrar imagen
float nuevaW = imagen.width * escala;  //Nuevas dimensiones
float nuevaH = imagen.height * escala;

float posX = (width - nuevaW) / 2;     //Posiciones posX y posY
float posY = (height - nuevaH) / 2;
```

3. Con estas correcciones se incorporaron dos transformaciones adicionales. La primera fue "***translate(posX, posY)***", que colocó el punto de inicio en la posición adecuada para que la imagen quede centrada. La segunda fue "***scale(escala)***", que redimensionó la imagen de acuerdo al factor proporcional calculado. Con estas modificaciones, la instrucción "***image(imagen, 0, 0)***" mostró la imagen ajustada al máximo tamaño posible dentro de la ventana y manteniendo siempre su proporción original.

```
  // Dibujar con transformaciones
  pushMatrix();
  translate(posX, posY);
  scale(escala);
  image(imagen, 0, 0);
  popMatrix();
}
```

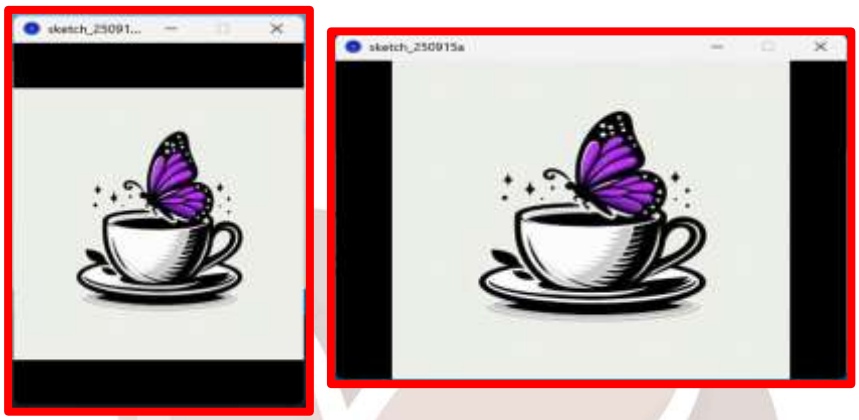
Materiales y equipos utilizados

- Laptop
- Processing
- Archivo PDF del profesor con información del tema

Resultados

Práctica 3-1

Se pudo apreciar la imagen cargada de forma clara.



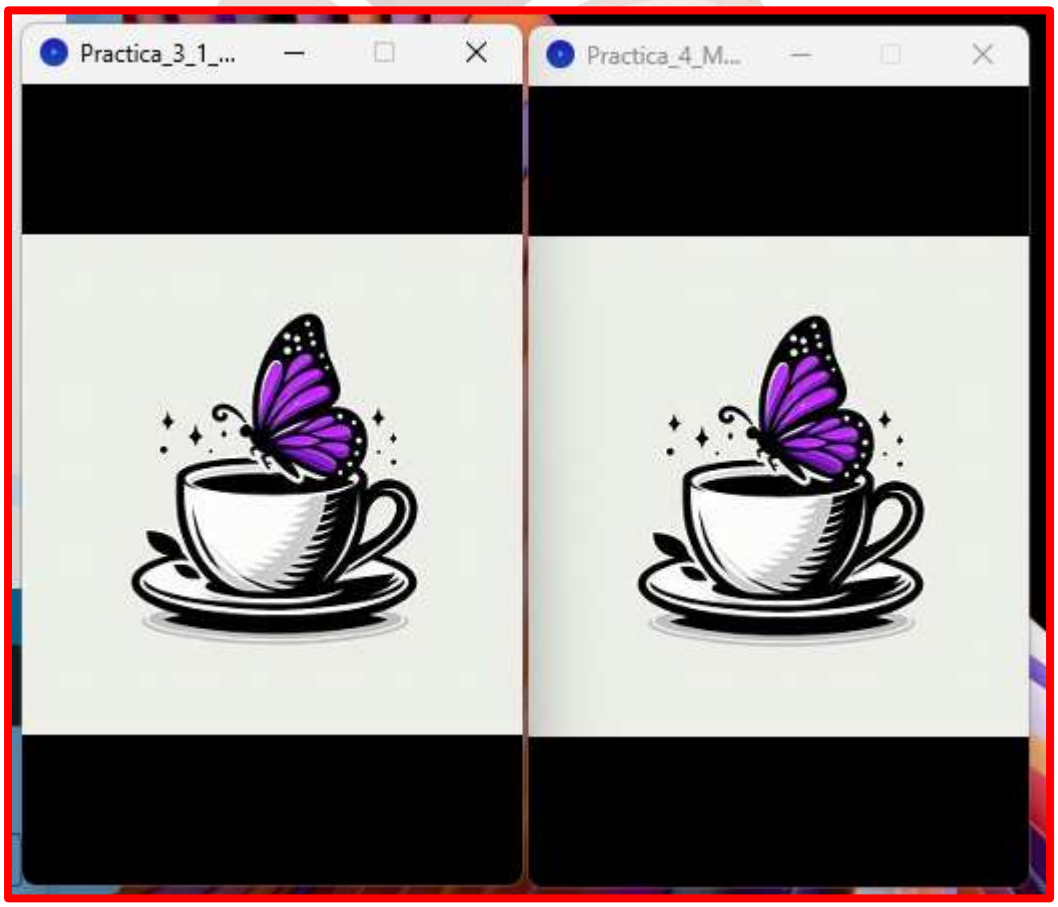
Práctica 3-2

El código mostro la imagen en una escala de grises y debajo el histograma.

Práctica 4

El código modificado de la practica 2-1 pudo mostrar las funciones seno y coseno sin problemas y de igual forma que el código original.

El código modificado de la practica 3-1 pudo mostrar la imagen cargada de la misma manera que el código original.



Análisis

La implementación de las prácticas 3 y 4 fue exitosa. En la práctica 3-1, se logró cargar y visualizar una imagen de forma clara, utilizando "***loadImage()***" e "***image()***". En la práctica 3-2, el código funcionó correctamente, mostrando la imagen en escala de grises y su histograma correspondiente en la parte inferior de la ventana, lo que demostró la capacidad de analizar la distribución de brillo de la imagen.

La práctica 4 demostró la superioridad de las transformaciones 2D sobre el mapeo manual de coordenadas. Al modificar la práctica 2-1, las funciones trigonométricas se pudieron dibujar correctamente al trasladar el origen y escalar los ejes con "**translate()**" y "***scale()***", eliminando la necesidad de conversiones manuales. Del mismo modo, en la modificación de la práctica 3-1, se logró centrar y redimensionar la imagen de forma proporcional usando "***translate()***" y "***scale()***", lo que ofreció una solución más flexible y robusta en comparación con el código original.

Discusión

Las prácticas abordaron dos aspectos esenciales de la graficación. La práctica 3 proporcionó una comprensión sólida de las operaciones de procesamiento de imágenes, como la carga, visualización y análisis de píxeles, siendo la visualización del histograma una herramienta valiosa para entender la distribución de intensidad en una imagen.

La práctica 4 puso de relieve la importancia y la elegancia de las transformaciones 2D como una alternativa a los cálculos manuales. La modificación de los códigos demostró que el uso de la matriz de modelo/vista en Processing es un enfoque mucho más eficiente para gestionar los cambios de coordenadas y el escalado. La implementación de "***pushMatrix()***" y "***popMatrix()***" también resaltó la importancia de poder aislar transformaciones sin que afecten al resto del programa, lo que mejora la estructura y la legibilidad del código.

Conclusión

Las prácticas 3 y 4 permitieron aplicar conceptos fundamentales de la graficación en Processing. Se logró manipular imágenes, desde su carga y visualización hasta el cálculo y representación de su histograma. Además, se comprobó que el uso de transformaciones 2D con las funciones "***translate()***" y "***scale()***" es un método más potente y versátil que los cálculos manuales para adaptar y centrar gráficos en la ventana. Estas prácticas ofrecieron un conocimiento práctico valioso sobre el manejo de elementos gráficos y la aplicación de transformaciones geométricas, consolidando una base fundamental para el desarrollo de proyectos más avanzados en graficación.

Referencias

- Linares i Pellicer, J. (s.f.). Gráficos por Computador: Imágenes y texto [documento PDF no publicado]. Escola Politècnica Superior d'Alcoi, Dep. de Sistemes Informàtics i Computació.
- Linares i Pellicer, J. (s.f.). Gráficos por Computador: Transformaciones 2D [documento PDF no publicado]. Escola Politècnica Superior d'Alcoi, Dep. de Sistemes Informàtics i Computació.