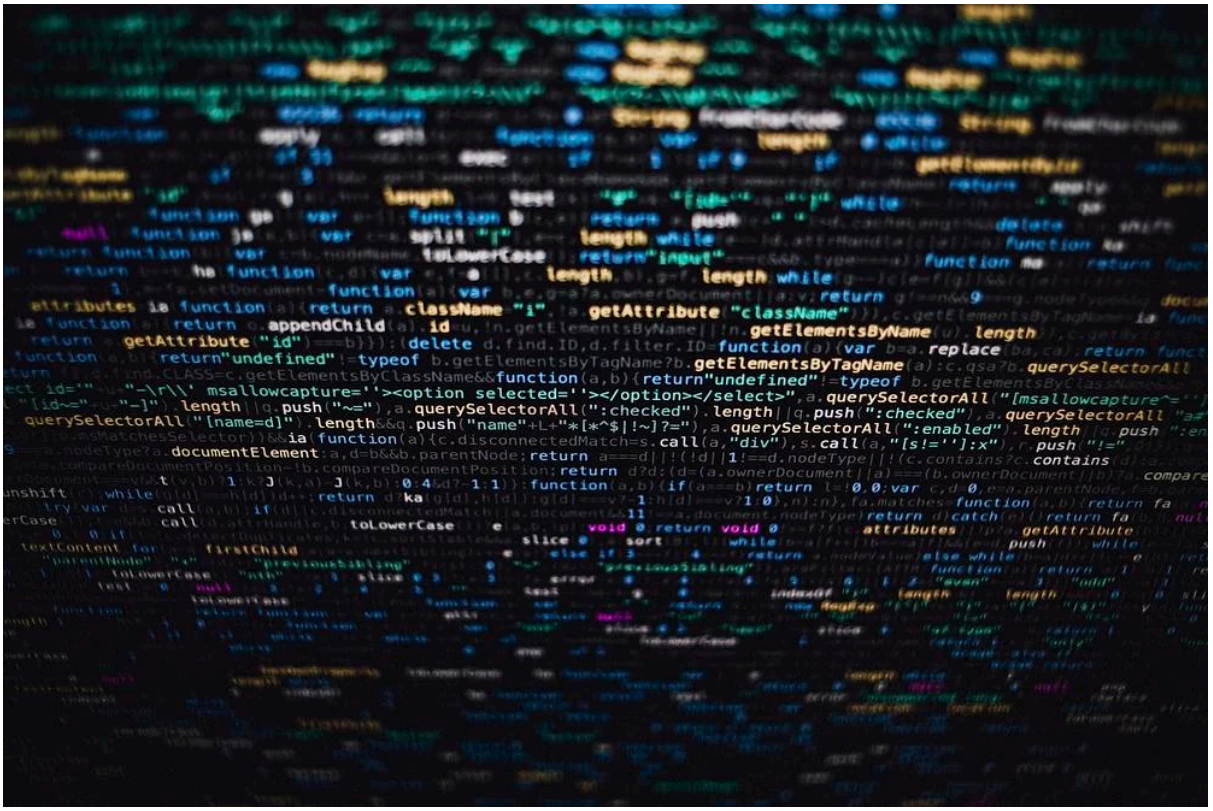
Reporte de la actividad del GIF

ENES
JURIQUILLA

Fecha de entrega: 7 de Octubre del 2024
Licenciatura en tecnología: 3er semestre
Asignatura: Programación orientada a objetos
Integrantes:

- Alicia Coronel Martínez
- Bryan Coronel Ramirez



Profesor: Ulises Olivares Pinto

Introducción

El formato de intercambio de gráficos, o mejor conocido por sus siglas en inglés GIF (Graphics Interchange Format) es un formato digital usado de forma muy común para el uso de animaciones.

Creado por la compañía CompuServe en 1987, los GIF ganaron mucha popularidad debido a su capacidad de ser comprimidos por el algoritmo de compresión LZW (por Lempel Ziv Welch), que permitia una descarga de velocidad razonable, incluso cuando la latencia era alta.

Sin embargo, los GIF cuentan con una limitante de que solo pueden ser usados por un byte de memoria, o lo que es igual a 8 bits. Esto significa que solo podemos usar hasta 256 colores.

Los GIF además tienen la ventaja de que no pierden calidad.

Los GIFs ganaron mucha popularidad debido a que fueron usados por el navegador Netscape. En 1988 CompuServe fue comprada por America Online, quien dejó expirar la patente de ellos, convirtiéndose entonces a un formato de acceso libre para el público en general.

Los GIF fueron aumentando y decayendo en popularidad, sin embargo cabe resaltar que grandes compañías como WhatsApp o Facebook la implementaron en sus plataformas, aun cuando desde el año 2000, con la invención de Flash, los GIF sufrieron una gran caída en popularidad.

Como diría Katherin Martin, al ver que en 2012 la palabra GIF se convirtió en palabra del año: “el GIF ya no es solamente un medio de expresión de la cultura pop: se ha convertido en una herramienta para la investigación y el periodismo, y su identidad léxica se transforma y se mantiene”.

Objetivos

- Reconocer las implicaciones gráficas del algoritmo de Ray Tracing.
- Explorar la posibilidad de representar al mundo físico mediante el uso de matrices.
- Desarrollar funciones que modifiquen matrices asociadas a objetos para cambiar la apariencia de susodichos objetos.

- Hipótesis

Con el uso de transformaciones sobre las matrices que representan la posición de los objetos podremos generar cambios en la apariencia de dichos objetos.

- Justificación

El uso de transformaciones lineales es un proceso lógico y con gran historia en el desarrollo de gráficos. Debería ser posible generar un GIF solamente con transformaciones lineales.

Trabajos relacionados:

En la Universidad Tecnológica Nacional de Buenos Aires existe un amplio repertorio de GIFs relacionados a funciones y comportamientos matemáticos. Denominado como "Los GIFs del material teórico" podemos ver una colección de hasta 23 GIFs diferentes que representan diversas gráficas con amplia presencia de vectores, planos y coordenadas. (*Los GIFs Del Material Teórico - Álgebra Y Geometría Analítica*, 2018)

Metodología
Librerías necesarias:

```
import numpy as np
import matplotlib.pyplot as plt
import imageio
```

Creación de la clase "Scene"

```
class Scene:
    def __init__(self, ancho, alto):
        self.ancho = ancho
        self.alto = alto
        # Definición de la imágen de salida (3 colores)
        self.imagen = np.zeros((self.alto, self.ancho, 3))
        # Inicializar cámara en una posición
        self.camara = np.array([0, 0, 1])
        # Proporción de la imágen
        prop = float(self.ancho / self.alto)
        # Posición de la pantalla (fustrum)
```

```
        self.pantalla = (-1, 1 / prop, 1, -1 / prop)  # izquirda,
arriba, derecha, abajo

    # Normalizar un vector
    def normalizar(self, vector):
        return vector / np.linalg.norm(vector)

    # Intersección entre un rayo y una esfera
    def intersecionEsfera(self, centro, radio, origen, direccion):
        b = 2 * np.dot(direccion, origen - centro)
        c = np.linalg.norm(origen - centro) ** 2 - radio ** 2
        delta = b ** 2 - 4 * c
        if delta > 0:
            t1 = (-b + np.sqrt(delta)) / 2
            t2 = (-b - np.sqrt(delta)) / 2
            if t1 > 0 and t2 > 0:
                return min(t1, t2)
        return None

    def intersecionMasCercana(self, objetos, origen, direccion):
        distancias = [self.intersecionEsfera(obj['centro'],
obj['radio'], origen, direccion) for obj in objetos]
        objetoCercano = None
        distanciaMin = np.inf
        for indice, distancia in enumerate(distancias):
            if distancia and distancia < distanciaMin:
                distanciaMin = distancia
                objetoCercano = objetos[indice]
        return objetoCercano, distanciaMin

    def reflected(self, vector, axis):
        return vector - 2 * np.dot(vector, axis) * axis

    def renderizarImagen(self, objetos, luz):
        print("Renderizando...")
        self.imagen.fill(0)  # Limpiar la imagen con negro (o cualquier
color de fondo)

        cont = 0
        for i, y in enumerate(np.linspace(self.pantalla[1],
self.pantalla[3], self.alto)):
            for j, x in enumerate(np.linspace(self.pantalla[0],
self.pantalla[2], self.ancho)):
```

```
                # definir pixel y origen
                pixel = np.array([x, y, 0])
                origen = self.camara
                direccion = self.normalizar(pixel - origen)
                reflexion = 1
                color = np.zeros((3))
                cont += 1
                if (((cont / (self.alto * self.ancho)) * 100) % 5 ==
0):
                    print((cont / (self.alto * self.ancho)) * 100, "%
completado")
                dir = self.normalizar(pixel - origen)
                objetoMasCercano, distanciaMin =
self.intersecionMasCercana(objetos, origen, dir)
                if objetoMasCercano is None:
                    continue
                interseccion = origen + distanciaMin * dir
                normalASuperficie = self.normalizar(interseccion -
objetoMasCercano['centro'])
                puntoDesplazado = interseccion + 1e-5 *
normalASuperficie
                interseccionConLuz = self.normalizar(luz['posicion'] -
puntoDesplazado)
                _, distanciaMin = self.intersecionMasCercana(objetos,
puntoDesplazado, interseccionConLuz)
                interseccionConLuzDistancia =
np.linalg.norm(luz['posicion'] - interseccion)
                esSombreado = distanciaMin <
interseccionConLuzDistancia
                if esSombreado:
                    continue
                iluminacion = np.zeros((3))
                iluminacion += objetoMasCercano['ambient'] *
luz['ambient']
                iluminacion += objetoMasCercano['diffuse'] *
luz['diffuse'] * np.dot(interseccionConLuz, normalASuperficie)
                interseccionConCam = self.normalizar(self.camara -
interseccion)
                H = self.normalizar(interseccionConLuz +
interseccionConCam)
                iluminacion += objetoMasCercano['specular'] *
luz['specular'] * np.dot(normalASuperficie, H) ** (
                        objetoMasCercano['shininess'] / 4)
```

```
                color += reflexion * iluminacion
                reflexion *= objetoMasCercano['reflection']
                origen = puntoDesplazado
                direccion = self.reflected(direccion,
normalASuperficie)
                self.imagen[i, j] = np.clip(color, 0, 1)

    def mostrarImg(self):
        plt.imshow(self.imagen)

    def guardarImg(self, nombre):
        plt.imsave(nombre, self.imagen)
        print(f"Imágen {nombre} generada correctamente!")
```

Creación de la clase “Transformaciones”

```
class Transformaciones:
    def __init__(self):
        pass

    def matriz_traslacion(self, tx, ty, tz):
        """Retorna una matriz de traslación 4x4"""
        return np.array([
            [1, 0, 0, tx],
            [0, 1, 0, ty],
            [0, 0, 1, tz],
            [0, 0, 0, 1]
        ])

    def matriz_rotacion_y(self, angulo):
        """Retorna una matriz de rotación alrededor del eje Y 4x4"""
        cos_ang = np.cos(np.radians(angulo))
        sin_ang = np.sin(np.radians(angulo))
        return np.array([
            [cos_ang, 0, sin_ang, 0],
            [0, 1, 0, 0],
            [-sin_ang, 0, cos_ang, 0],
            [0, 0, 0, 1]
        ])

    def matriz_escalado(self, sx, sy, sz):
        """Retorna una matriz de escalado 4x4"""
        return np.array([
```

```
            [sx, 0, 0, 0],
            [0, sy, 0, 0],
            [0, 0, sz, 0],
            [0, 0, 0, 1]
        ])

    def aplicar_transformacion(self, objeto, transformacion):
        """Aplica una transformación a un objeto"""
        centro_homogeneo1 = np.append(objeto['centro'], 1)  # Convertir
a coordenadas homogéneas (x, y, z, 1)
        centro_homogeneo = np.transpose(centro_homogeneo1)
        nuevo_centro = np.dot(transformacion, centro_homogeneo)  #
Multiplicación de matrices
        objeto['centro'] = nuevo_centro[:3]  # Volver a convertir a
coordenadas 3D (x, y, z)

    def traslacion(self, objeto, tx, ty, tz):
        """Aplica una traslación a un objeto"""
        T = self.matriz_traslacion(tx, ty, tz)
        self.aplicar_transformacion(objeto, T)

    def rotacion(self, objeto, angulo):
        """Aplica una rotación a un objeto alrededor del eje Y"""
        R = self.matriz_rotacion_y(angulo)
        self.aplicar_transformacion(objeto, R)

    def escalado(self, objeto, sx, sy, sz):
        """Aplica un escalado a un objeto"""
        S = self.matriz_escalado(sx, sy, sz)
        self.aplicar_transformacion(objeto, S)
```

Creación de la función “animar_escena”

```
def animar_escena(objScene, transformaciones, frames=60):
    objetos = [
        {'centro': np.array([0, -.10, 0]), 'radio': 9000 - 0.7,
'ambient': np.array([0.1, 0.1, 0.1]), 'diffuse': np.array([0.6, 0.6,
0.6]), 'specular': np.array([1, 1, 1]), 'shininess': 100, 'reflection':
0.5},
```

```
        {'centro': np.array([-0.5, 0, -1]), 'radio': 0.7, 'ambient':
np.array([0.1, 0, 0]), 'diffuse': np.array([0.7, 0, 0]), 'specular':
np.array([1, 1, 1]), 'shininess': 100, 'reflection': 0.5},
        {'centro': np.array([0.1, -0.3, 0]), 'radio': 0.1, 'ambient':
np.array([0.1, 0, 0.1]), 'diffuse': np.array([0.7, 0, 0.7]),
'specular': np.array([1, 1, 1]), 'shininess': 100, 'reflection': 0.5},
        {'centro': np.array([-0.3, 0, 0]), 'radio': 0.15, 'ambient':
np.array([0, 0.1, 0.1]), 'diffuse': np.array([0, 0.6, 0.6]),
'specular': np.array([1, 1, 1]), 'shininess': 100, 'reflection': 0.5}
    ]

    luz = {
        'posicion': np.array([5, 5, 5]),
        'ambient': np.array([1, 1, 1]),
        'diffuse': np.array([1, 1, 1]),
        'specular': np.array([1, 1, 1])
    }

    for frame in range(frames):
        print(f"Generando frame {frame + 1}/{frames}")
        # Aplicar rotación a los objetos en cada frame
        transformaciones.traslacion(objetos[1], 0.01, 0, 0)  #
Trasladar el objeto[1]
        transformaciones.traslacion(objetos[2], -0.01, 0, 0)  #
Trasladar el objeto[2]
        transformaciones.escalado(objetos[3], 1.01, 1.01, 1.01)  #
Escalar el objeto[3]
        transformaciones.escalado(objetos[2], 1.01, 1.01, 1.01)  #
Escalar el objeto[2]
        transformaciones.escalado(objetos[2], .9, .9, .9)  # Escalar el
objeto[1]
        transformaciones.rotacion(objetos[1], 5)  # Rotar el objeto 1
ligeramente en cada frame
        transformaciones.rotacion(objetos[2], -3)  # Rotar el objeto 2
en dirección contraria
        objScene.camara = (0.0001*frame, 0.0001*frame, 1 + 0.1*frame)


        objScene.renderizarImagen(objetos, luz)
        objScene.mostrarImg()
        objScene.guardarImg(f"frame_{frame}.png")
```

Creación de la escena y aplicar las transformaciones.

```
# Crear la escena y aplicar las transformaciones
ancho, alto = 400, 400
escena = Scene(ancho, alto)
transformaciones = Transformaciones()
```

Generación de la animación.

```
animar_escena(escena, transformaciones, frames=20)


img_array = []
frames = 90
# Leer todos los archivos formato imagen desde path
for frame in range(frames):



   # Asignar a variable leer_imagen, el nombre de cada imagen
   leer_imagen = imageio.v2.imread(f"frame_{frame}.png")

   # añadir imágenes al arreglo img_array
   img_array.append(leer_imagen)

# Guardar Gif
imageio.mimwrite('Gato.gif', img_array, 'GIF', fps = 30, duration = 3)
```

Resultados:

Las imágenes se pudieron generar correctamente, cada una con una diferencia directamente asociada a la función que se le aplicó a la misma.

Conclusión:

El uso de transformaciones sobre las matrices relacionadas a las posiciones de los objetos genera cambios cuando a estos objetos se les toma una “captura”, demostrando que efectivamente dichos cambios fueron realizados.

Esto significa que nuestras funciones de transformación son correctas y también modulares; se puede rehusar el código cuando se convenga.

Referencias

[0] colaboradores de Wikipedia. (2024, January 24). *Graphics Interchange format*. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Graphics_Interchange_Format

[1] *Why does a byte only have 0 to 255?* (n.d.). Stack Overflow. https://stackoverflow.com/questions/4986486/why-does-a-byte-only-have-0-to-255

[2] *Los GIFs del material teórico - Álgebra y Geometría Analítica*. (2018, October 17). Álgebra Y Geometría Analítica. https://aga.frba.utn.edu.ar/gifs-material-teorico/

[3] Kipuna. (2024, 15 abril). Crear un GIF con imágenes – python imageio - Kipuna Ec. *Kipuna Ec*. https://kipunaec.com/crear-un-gif-con-imagenes-python-imageio/