
	<b>VICERRECTORADO DOCENTE</b>	<b>Código:</b> GUIA-PRL-001
	CONSEJO ACADÉMICO	<b>Aprobación:</b> 2016/04/06
<b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

		<b>PRÁCTICA DE LABORATORIO</b>	
<b>CARRERA:</b> Ingeniería en Ciencias de la Computacion		<b>ASIGNATURA:</b> Computacion Paralela	
<b>NRO. PRÁCTICA:</b>	5	<b>TÍTULO PRÁCTICA:</b> Examen Interciclo	
<b>OBJETIVO ALCANZADO:</b> Consumir la aplicación dockerizada mediante una aplicación de android			
<b>ACTIVIDADES DESARROLLADAS</b>			
<b>1. Seleccionar tres filtros para aplicar a imágenes</b>  <b>a) Anamorphic</b> Modifica las proporciones de la imagen, estirándola horizontalmente para replicar el efecto de las lentes anamórficas utilizadas en cine, lo que proporciona una apariencia más amplia y cinematográfica. <b>b) Sepia</b> Aplica un tono marrón cálido uniforme a toda la imagen, imitando el aspecto de las fotografías antiguas y dándole un aire nostálgico o retro. <b>c) Sobremontado de imagen en el centro de otra</b> Superpone múltiples imágenes o elementos gráficos sobre una imagen principal, creando un efecto visual compuesto que puede añadir profundidad, contexto o un diseño artístico más complejo.			
<b>2. Código utilizado para aplicar los filtros</b> <b>a) APP.PY</b> <pre> from flask import Flask import os  import logging  import firebase_admin  from firebase_admin import credentials, firestore from flask_jwt_extended import JWTManager  # Configuración de logging logging.basicConfig(level=logging.DEBUG) </pre>			

```
# Inicializar Firebase Admin
cred = credentials.Certificate('proyectoFlask-47f4f-firebase-adminsdk-l8uq4-3b36e0d55f.json')
firebase_admin.initialize_app(cred)

# Obtén una referencia a la base de datos Firestore
db = firestore.client()

def create_app():
    app = Flask(__name__)

    # Configurar la clave secreta para JWT
    app.config['JWT_SECRET_KEY'] = os.getenv('JWT_SECRET_KEY', 'password')

    # Inicializar JWT Manager
    jwt = JWTManager(app)

    # Importar y registrar los blueprints
    from .views import app_views # Importación relativa porque `views.py` está en el mismo
    directorio
    app.register_blueprint(app_views, url_prefix='/api')

    return app

# Crear instancia de la aplicación
app = create_app()

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

**b) Filters.py**

```
import os
from PIL import Image
import numpy as np
import pycuda.driver as cuda
from pycuda.compiler import SourceModule

def apply_sepia(input_image, intensity=2.0, add_noise=True, vignette=True):
    input_array = np.array(input_image).astype(np.float32) / 255.0
    output_array = np.zeros_like(input_array)

    cuda.init()
    device = cuda.Device(0)
    context = device.make_context()

    try:
        mod = SourceModule("""
            _global_ void sepia_kernel(float *d_image, float *d_result, int width, int height,
            float intensity) {

                int x = threadIdx.x + blockIdx.x * blockDim.x;
                int y = threadIdx.y + blockIdx.y * blockDim.y;
                int idx = (y * width + x) * 3;
                if (x < width && y < height) {
                    float r = d_image[idx];
                    float g = d_image[idx + 1];
                    float b = d_image[idx + 2];
                    float new_r = min(intensity * (0.393f * r + 0.769f * g + 0.189f * b), 1.0f);
```

```
        float new_g = min(intensity * (0.349f * r + 0.686f * g + 0.168f * b), 1.0f);
        float new_b = min(intensity * (0.272f * r + 0.534f * g + 0.131f * b), 1.0f);

        d_result[idx] = new_r;
        d_result[idx + 1] = new_g;
        d_result[idx + 2] = new_b;
    }
}
""")

sepia_kernel = mod.get_function("sepia_kernel")
sepia_kernel(
    cuda.In(input_array), cuda.Out(output_array),
    np.int32(input_image.width), np.int32(input_image.height), np.float32(intensity),
    block=(16, 16, 1), grid=(int((input_image.width + 15) // 16),
int((input_image.height + 15) // 16))
)

if add_noise:
    noise = np.random.normal(loc=0.0, scale=0.05,
size=output_array.shape).astype(np.float32)
    output_array += noise

if vignette:
    center_x, center_y = input_image.width / 2, input_image.height / 2
    max_radius = np.sqrt(center_x*2 + center_y*2)
    for y in range(input_image.height):
        for x in range(input_image.width):
            radius = np.sqrt((x - center_x) * 2 + (y - center_y) * 2)
            scale = radius / max_radius
            output_array[y, x] *= (1 - scale * 0.5)
```

```
        output_image = Image.fromarray(np.uint8(output_array * 255))

        return output_image

    finally:
        context.pop()

def apply_anamorphic(input_image):
    input_array = np.array(input_image).astype(np.float32) / 255.0
    output_array = np.zeros_like(input_array)

    cuda.init()
    device = cuda.Device(0)
    context = device.make_context()

    try:
        mod = SourceModule("""
__global__ void anamorphic_kernel(float *d_image, float *d_result, int width, int
height) {

    int x = threadIdx.x + blockIdx.x * blockDim.x;
    int y = threadIdx.y + blockIdx.y * blockDim.y;
    int idx = (y * width + x) * 3;

    float brightness = 0.299f * d_image[idx] + 0.587f * d_image[idx + 1] + 0.114f *
d_image[idx + 2];

    if (brightness > 0.55) { // Only apply effect to bright areas
        float effect_strength = 1.5;
        int spread = 25; // Spread effect to 25 pixels on either side
        for (int i = -spread; i <= spread; i++) {
            int new_x = x + i;

            if (new_x >= 0 && new_x < width) {
                int new_idx = (y * width + new_x) * 3;
```

```
        atomicAdd(&d_result[new_idx], d_image[idx] * effect_strength / (spread
* 2));

        atomicAdd(&d_result[new_idx + 1], d_image[idx + 1] * effect_strength /
(spread * 2));

        atomicAdd(&d_result[new_idx + 2], d_image[idx + 2] * effect_strength /
(spread * 2));

    }

}

} else {

    d_result[idx] = d_image[idx];
    d_result[idx + 1] = d_image[idx + 1];
    d_result[idx + 2] = d_image[idx + 2];

}

}

""")

anamorphic_kernel = mod.get_function("anamorphic_kernel")
anamorphic_kernel(

    cuda.In(input_array), cuda.Out(output_array),

    np.int32(input_image.width), np.int32(input_image.height),

    block=(16, 16, 1), grid=(int((input_image.width + 15) // 16),
int((input_image.height + 15) // 16))

)

output_image = Image.fromarray(np.uint8(output_array * 255))

return output_image

finally:

    context.pop()

import pycuda.autoinit
import pycuda.driver as cuda
import numpy as np
```

```
from pycuda.compiler import SourceModule
from PIL import Image
import os

def apply_logo(input_image):
    input_array = np.array(input_image).astype(np.float32) / 255.0
    logo_path = os.path.join(os.path.dirname(os.path.dirname(__file__)), 'static',
                              'logoups.jpeg')
    logo_image = Image.open(logo_path).convert('RGBA').resize((150, 150))
    logo_array = np.array(logo_image).astype(np.float32) / 255.0

    output_array = np.zeros_like(input_array)

    cuda.init()
    device = cuda.Device(0)
    context = device.make_context()

    try:
        mod = SourceModule("""
            __global__ void logo_kernel(float *d_image, float *d_logo, float *d_result, int
img_width, int img_height, int logo_width, int logo_height) {
                int x = threadIdx.x + blockIdx.x * blockDim.x;
                int y = threadIdx.y + blockIdx.y * blockDim.y;

                if (x < img_width && y < img_height) {
                    int idx = (y * img_width + x) * 3;
                    d_result[idx] = d_image[idx]; // Copy RGB directly
                    d_result[idx + 1] = d_image[idx + 1];
                    d_result[idx + 2] = d_image[idx + 2];
                }
            }
        """)
```

```
// Coordinates to center the logo

int cx = (img_width - logo_width) / 2;
int cy = (img_height - logo_height) / 2;

if (x >= cx && x < (cx + logo_width) && y >= cy && y < (cy + logo_height)) {
    int logo_idx = ((y - cy) * logo_width + (x - cx)) * 4; // Logo has alpha
channel
    float alpha = d_logo[logo_idx + 3];
    // Blend with alpha
    d_result[idx] = alpha * d_logo[logo_idx] + (1 - alpha) * d_result[idx];
    d_result[idx + 1] = alpha * d_logo[logo_idx + 1] + (1 - alpha) *
d_result[idx + 1];
    d_result[idx + 2] = alpha * d_logo[logo_idx + 2] + (1 - alpha) *
d_result[idx + 2];
    }
}

}

""")

logo_kernel = mod.get_function("logo_kernel")
logo_kernel(
    cuda.In(input_array), cuda.In(logo_array), cuda.Out(output_array),
    np.int32(input_image.width), np.int32(input_image.height),
    np.int32(logo_image.width), np.int32(logo_image.height),
    block=(16, 16, 1), grid=(int((input_image.width + 15) // 16),
int((input_image.height + 15) // 16))
)

output_image = Image.fromarray(np.uint8(output_array * 255))
return output_image

finally:
```



```
context.pop()

def apply_filter(image, filter_type):
    if filter_type == 'sepia':
        return apply_sepia(image)
    elif filter_type == 'anamorphic':
        return apply_anamorphic(image)
    elif filter_type == 'logo':
        return apply_logo(image)
    else:
        raise ValueError("Unknown filter type")
```

### c) VIEWS.PY

```
from flask import Blueprint, request, jsonify
from werkzeug.security import generate_password_hash, check_password_hash
from flask_jwt_extended import create_access_token, jwt_required, get_jwt_identity
from PIL import Image
import io
import firebase_admin
from firebase_admin import firestore
from .filters import apply_filter

db = firestore.client()
app_views = Blueprint('app_views', __name__)

@app_views.route('/upload', methods=['POST'])
def upload_image():
    filter_type = request.form.get('filter', 'sepia')
```

```
image_file = request.files['image']
if not image_file:
    return jsonify({'error': 'No image file provided'}), 400

image = Image.open(image_file.stream)
filtered_image = apply_filter(image, filter_type)
img_byte_arr = io.BytesIO()
filtered_image.save(img_byte_arr, format='JPEG')
img_byte_arr.seek(0)

file_path = f'static/processed_{filter_type}.jpg'
filtered_image.save(file_path)

new_image_ref = db.collection('processed_images').document()
new_image_ref.set({
    'filter_type': filter_type,
    'image_path': file_path
})

return jsonify({'message': 'Image uploaded and processed successfully', 'image_url':
file_path}), 201

@app_views.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    email = data.get('email')
    password = data.get('password')
    if not email or not password:
        return jsonify({"message": "Email and password required"}), 400
```

```
users_ref = db.collection('users')

user_query = users_ref.where('email', '==', email).get()

if list(user_query):

    return jsonify({"message": "User already exists"}), 409

password_hash = generate_password_hash(password)

new_user_ref = users_ref.document()

new_user_ref.set({

    'email': email,

    'password_hash': password_hash

})

return jsonify({"message": "User created successfully"}), 201

@app_views.route('/login', methods=['POST'])
def login():

    data = request.get_json()

    email = data.get('email')

    password = data.get('password')

    if not email or not password:

        return jsonify({"message": "Email and password required"}), 400

    users_ref = db.collection('users')

    user_query = users_ref.where('email', '==', email).get()

    if not list(user_query) or not check_password_hash(list(user_query)[0].to_dict()['password_hash'], password):

        return jsonify({"message": "Invalid credentials"}), 401

    # Create a new access token for the authenticated user
```

```
access_token = create_access_token(identity=email)

return jsonify({"message": "Login successful", "access_token": access_token}), 200

@app_views.route('/users', methods=['GET'])
def get_users():
    users_ref = db.collection('users')
    docs = users_ref.stream()
    users_list = [doc.to_dict() for doc in docs]
    return jsonify(users_list), 200
```

CODIGO EN ANDROID STUDIO

### 3. LINKS DEL PROYECTO EN GITHUB

[https://github.com/ChristianJapon/API\\_Mobil.git](https://github.com/ChristianJapon/API_Mobil.git)

#### 4. Resultados obtenidos aplicando los filtros

##### a) Resultado filtro ANAMORPHIC





**b) Resultado filtro SEPIA**




**c) Resultado FILTRO SOBREMONTADO DE IMÁGENES**



**RESULTADO(S) OBTENIDO(S):**

Hemos aplicado el filtro de manera exitosa y se pudo observar que los filtros aplicados con pycuda y este a su vez que fue dockerizado si es usado de manera eficiente y cumple con el proposito para el cual fue programado

	<b>VICERRECTORADO DOCENTE</b>	<b>Código:</b> GUIA-PRL-001
	CONSEJO ACADÉMICO	<b>Aprobación:</b> 2016/04/06
<b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

### CONCLUSIONES:

La aplicación esta basada en pycuda, estos filtros estan considerados para trabajar con bloques e hilos que nos permite ir aplicando en los distintos pixeles el filtro seleccionado, la dockerizacion nos permite habilitar el puerto que vamos a usar en este caso el 5000, en este esta funcionando nuestras APIs que seran utilizadas en nuestra aplicación de android(que funciona como Front End)

**Nombre de estudiante:** Ronald Andrade, Christian Japon

**Firma de estudiante:** \_\_\_\_\_