CC3501: Tarea 3A - Efecto Bloom

Camilo Núñez Barra

7 de enero de 2021

Resumen

El efecto bloom se utiliza para representar de manera relativamente realista y barata la difusión de la luz. Esta tarea implementa el efecto bloom en Python, resolviendo $\nabla^2 u = 0$, la EDP que recorre bidimensionalmente la luz en una imagen, utilizando el método de diferencias finitas, con ayuda de matrices *sparse* para ahorrar memoria. Se permite al usuario ingresar una imagen con un valor RGB específico de luz que se difumine en el rango que desee.

1. Solución propuesta

Una imagen tiene una resolución de $H \times W$ píxeles, con tres canales R, G y B para representar, respectivamente, el color rojo, verde y azul, adicionalmente puede tener un canal a para representar la transparencia. Estos canales toman valores enteros no negativos de 8 bits: un 0 significa que tal canal no aporta al color final del píxel, luego va aumentando el valor a la vez que aumenta el aporte al color final, hasta llegar al límite 255 con una intensidad máxima.

El color de la luz en una imagen tiene generalmente los tres canales de color con valores cercanos al máximo, donde RGB(255,255,255) corresponde al color blanco.

El efecto bloom difumina la luz de una imagen. Esta tarea considera que la luz sigue la ecuación de Laplace, dada por $\nabla^2 u = 0$, es decir, en una imagen bidimensional, $\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 = 0$. Esta es una ecuación diferencial parcial elíptica en estado estacionario, con condición de borde de Dirichlet dada por $u = \text{RGB(luz)}$ (ver línea 25) en cada píxel que corresponda al color de la luz que se quiere difuminar (a priori, ubicados en cualquier parte de la imagen, no necesariamente en el *borde*). Además, el dominio de la luz está dado por todos los píxeles dentro de un radio n de cercanía con los píxeles de luz (los de la condición de borde, exclusive), es decir, píxeles (x_p, y_p) tales que para cada luz (x_l, y_l) se tiene $(x_p - x_l)^2 + (y_p - y_l)^2 \leq n^2$ (ver línea 31). Todos los píxeles fuera de estas regiones no forman parte de la difuminación, así como tampoco la luz misma.

Gracias al Teorema de Taylor se obtiene la fórmula de diferencias centradas $O(h^2)$ para derivadas de segundo orden, $f^{(2)}(x_0) \approx (f_1 - 2f_0 + f_{-1})/h^2$, de modo que la ecuación de Laplace se puede discretizar mediante el método de diferencias finitas, obteniéndose la siguiente fórmula,

$$\nabla^2 u_{i,j} \approx \frac{u_{i-1,j} + u_{i+1,j} - 4u_{i,j} + u_{i,j-1} + u_{i,j+1}}{h^2} = 0.$$

El dominio está compuesto por píxeles, unidades elementales, por lo tanto, se escoge $h = 1$, la unidad. El brillo extra para un pixel en la posición (i, j) es $u_{i,j}$, y, gracias

a la fórmula anterior, se puede expresar en función de sus cuatro pixeles contiguos: el superior, inferior, izquierdo y derecho, cuyos brillos extras son, respectivamente, $u_{i,j+1}$, $u_{i,j-1}$, $u_{i-1,j}$ y $u_{i+1,j}$, tal como se representa en la figura 1. Sabiendo de antemano que la coordenada (i, j) es una incógnita con brillo $u_{i,j}$ por determinar, se puede reconocer que cada vecino puede pertenecer a solo uno de los siguientes cuatro grupos: incógnitas u, condiciones de borde (luz), pixel fuera de rango n o pixel inválido fuera de la imagen (ver líneas 43–60). Luego, existen $4^4 = 256$ casos posibles para considerar todas las permutaciones posibles de vecinos.

Se buscan las coordenadas de todas las incógnitas de brillo extra u (ver línea 33), almacenándolas convenientemente en un diccionario. Se sabe que un pixel vecino pertenece al primer grupo si sus coordenadas pertecenen al diccionario (ver línea 62). Se sabe que un pixel pertenece al segundo grupo si corresponde a una condición de borde de Dirichlet (ver línea 53). Se sabe que un pixel pertenece al tercer grupo si no es una condición de borde de Dirichlet y sus índices son válidos dentro de la imagen (ver línea 53). Se sabe que un pixel pertenece al cuarto grupo si los índices no son válidos para la imagen (ver línea 54).

Los coeficientes en el laplaciano de las incógnitas de brillo u se almacenan en una matriz *sparse* denotada A (ver línea 37). Se elige esta estructura de datos pues existen a lo más cinco elementos no nulos por fila, un -4 (ver línea 67) para la incógnita misma y un 1 (ver líneas 63–66) por cada pixel vecino que también sea incógnita, es decir, para imágenes de resoluciones muy grandes, una matriz común desperdicia mucha memoria. Además, un vector b (ver línea 38) almacena las condiciones de borde de Dirichlet según corresponda a cada incógnita (ver línea 68). Sea x el vector de incógnitas, este se puede determinar al resolver el sistema de ecuaciones lineales $Ax = b$ (ver línea 71). Esto se realiza para cada canal de color análogamente.

El vector x contiene cada brillo extra u de los pixeles dentro del rango de difuminación, por lo tanto, a cada uno de los colores RGB originales de estos pixeles se les adiciona el brillo extra u (ver línea 74), y se debe asegurar que esto no sobrepase los límites del rango de valores de los canales.

2. Instrucciones de ejecución

1. Descargar Python 3.8.5 de https://www.python.org/downloads/release/python-385/ e instalar, añadiendo a PATH.

2. Descargar la carpeta `tarea3a` de https://github.com/camilo-nb/CC3501-tareas. La estructura de esta carpeta se muestra en la figura 2.

3. Abrir el intérprete de comandos en la carpeta `tarea3a`. Por ejemplo, en Microsoft Windows se puede usar `cmd.exe`.

4. Instalar las librerías: `numpy==1.19.1`, `Pillow==1.7.0` y `scipy==0.16.0`, mediante el comando `$ pip install -r requirements.txt`.

5. Ejecutar el código mediante el comando `$ python bloom_effect.py image_filename N R G B`. Los parámetros son:

   str – `image_filename` — nombre del archivo de imagen para aplicar efecto bloom. Se admiten solamente extensiones `.png` y `.jpg` (deben ir en la llamada).

uint – `N` — cantidad de píxeles de alcance (radio) que tiene la difusión.

8bit uint – `R` — valor del canal rojo del color a difuminar.

8bit uint – `G` — valor del canal verde del color a difuminar.

8bit uint – `B` — valor del canal azul del color a difuminar.

3. Resultados

La figura 3 muestra la ejecución de la instrucción 5, que permite aplicar el efecto bloom a la imagen presente en la figura 4. Se aprecia una correcta difusión de la luz, con los discos claramente definidos

Las figuras 4–11 muestran la aplicación del efecto a variadas imágenes, estableciendo como color de difusión al blanco puro RGB(255,255,255). Todas estas imágenes presentan una difusión esperable de la luz, sin embargo, la figura 11 solamente difumina la luz de ciertos objetos blancos, pero no todos, esto se debe netamente a que el programa está diseñado para difuminar exactamente el valor RGB de la llamada, sin considerar rangos de valores para permitir pequeños errores o variaciones de color, por lo que, por ejemplo, un casi-blanco RGB(255,255,254) no se toma en cuenta.

Las figuras 12–16 muestran un comportamiento no deseable del efecto, sin embargo, es completamente sensato desde la definición del efecto bloom: este difumina *luz* incrementando los valores de los colores vecinos, por lo que ingresar colores no claros o que no corresponden a luz, como, por ejemplo, azul, naranjo, rosado, café o negro, ilumina los vecinos proporcionalmente a valor RGB ingresado, sin incrementar tal valor RGB. Luego, los ojos del personaje de la figura 12, que tienen un gradiente de azul en el borde, se iluminan solo fuera del centro, causando una clara diferencia entre las zonas iluminadas y el color RGB de la llamada. Además, como la cantidad iluminación es proporcional al valor RGB de la llamada, en la figura 16 (un pixel negro) efectivamente no se difumina el color negro, pues este color corresponde a la ausencia de *luz*.

El video de demostración se puede ver en https://youtu.be/NhDfVV7sF2U.

Anexo

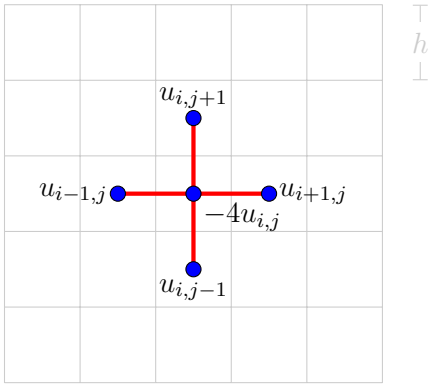


Figura 1: Esténcil de cinco puntos

```
tarea3a/
├── examples/
│   ├── colorwheel.png
│   ├── dices.png
│   ├── door.jpg
│   ├── dot_black.jpg
│   ├── dot_white.jpg
│   ├── elephants.png
│   ├── guy.png
│   ├── link.jpg
│   ├── link.png
│   └── matchstick.jpg
├── bloom_effect.py
├── demo.mp4
├── report.pdf
└── requirements.txt
```

Figura 2: Estructura de la carpeta `tarea3a`

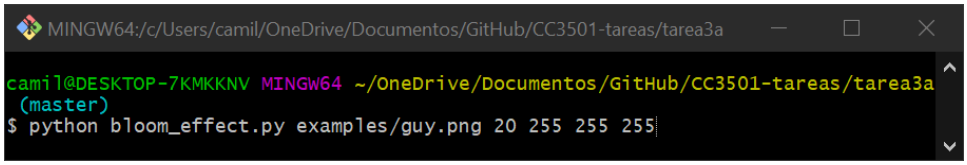


Figura 3: Ejecución del código en el intérprete de comándos

(a) Imagen original

(b) Efecto bloom

Figura 4: Efecto aplicado a `guy.png` (resolución 300×300). Parámetros: `N=20 RGB(255,255,255)`

(a) Imagen original

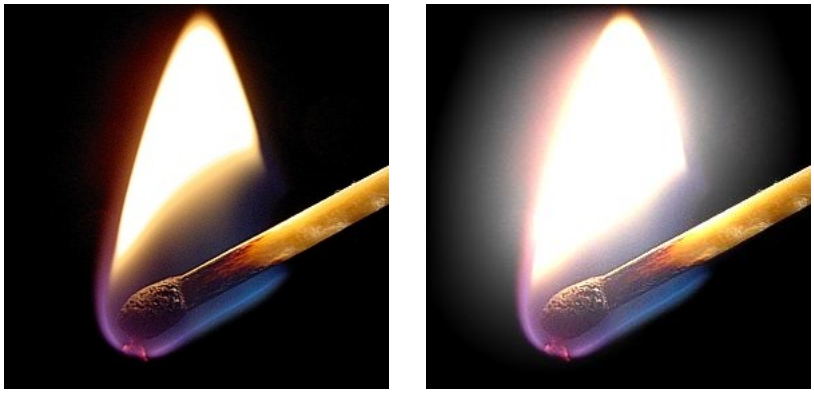
(b) Efecto bloom

Figura 5: Efecto aplicado a `link.jpg` (resolución 300×300). Parámetros: `N=50 RGB(255,255,255)`

(a) Imagen original

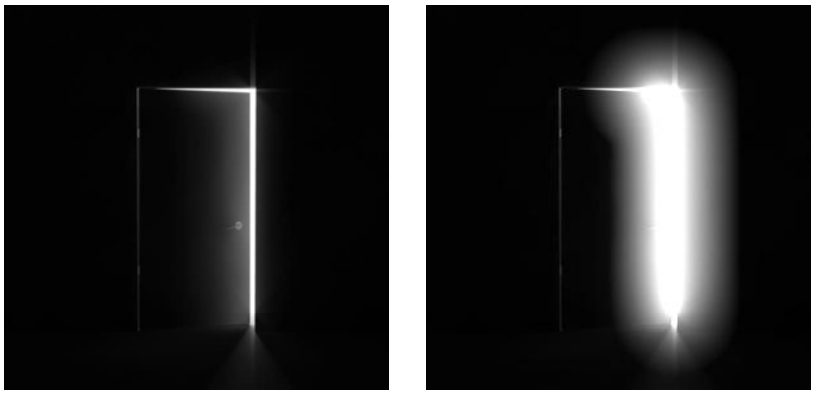
(b) Efecto bloom

Figura 6: Efecto aplicado a `dices.jpg` (resolución 800×600). Parámetros: `N=20 RGB(255,255,255)`



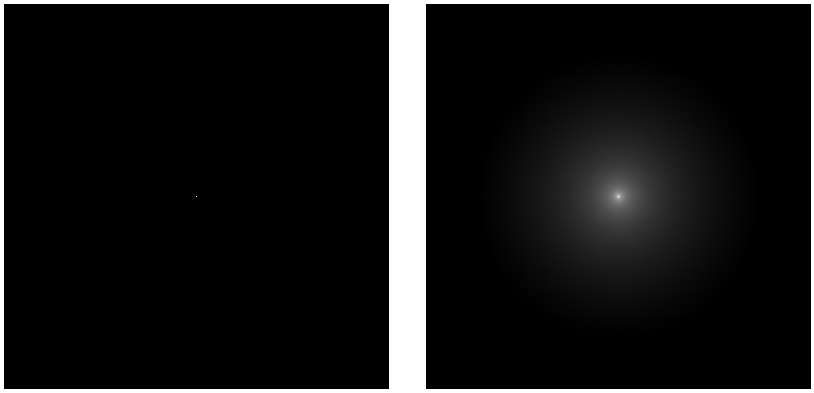
(a) Imagen original (b) Efecto bloom

Figura 7: Efecto aplicado a `matchstick.jpg` (resolución 288 × 288). Parámetros: `N=100 RGB(255,255,255)`



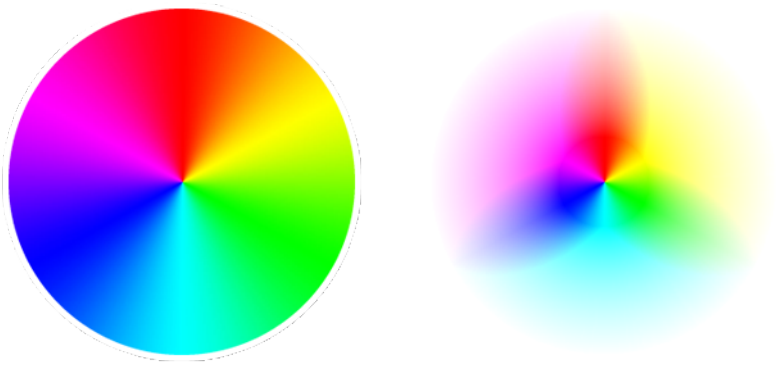
(a) Imagen original (b) Efecto bloom

Figura 8: Efecto aplicado a `door.jpg` (resolución 300 × 300). Parámetros: `N=50 RGB(255,255,255)`



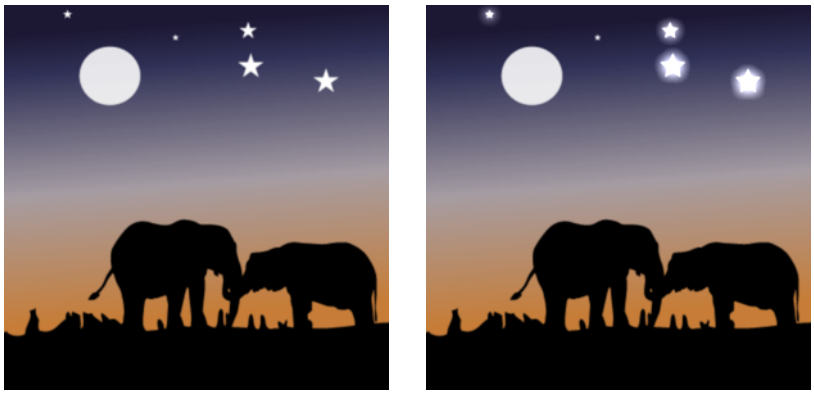
(a) Imagen original (b) Efecto bloom

Figura 9: Efecto aplicado a `dot_white.png` (resolución 401×401). Parámetros: `N=150 RGB(255,255,255)`



(a) Imagen original (b) Efecto bloom

Figura 10: Efecto aplicado a `colorwheel.png` (resolución 300 × 300). Parámetros: `N=100 RGB(255,255,255)`



(a) Imagen original (b) Efecto bloom

Figura 11: Efecto aplicado a `elephants.png` (resolución 300 × 300). Parámetros: `N=10 RGB(255,255,255)`

(a) Imagen original (b) Efecto bloom

Figura 12: Efecto aplicado a `link.png` (resolución 256 × 256). Parámetros: `N=100 RGB(30,146,203)`

(a) Imagen original

(b) Efecto bloom

Figura 13: Efecto aplicado a `link.png` (resolución 256×256). Parámetros: `N=100 RGB(255,98,25)`

(a) Imagen original

(b) Efecto bloom

Figura 14: Efecto aplicado a `guy.png` (resolución 300×300). Parámetros: `N=100 RGB(237,33,124)`

(a) Imagen original

(b) Efecto bloom

Figura 15: Efecto aplicado a `guy.png` (resolución 300×300). Parámetros: `N=25 RGB(59,35,19)`

(a) Imagen original (b) Efecto bloom

Figura 16: Efecto aplicado a `dot_black.png` (resolución 401 × 401). Parámetros: `N=150 RGB(0,0,0)`

Código 1: `bloom_effect.py`

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
import sys

import numpy as np
from PIL import Image
from scipy.sparse import dok_matrix
from scipy.sparse.linalg import spsolve

__author__ = "Camilo Núñez Barra"

# Handling input
if len(sys.argv) != 6: raise Exception("Invalid length of arguments. Follow:\n$
    ↪ python bloom_effect.py image_filename N R G B")
im_filename = sys.argv[1]
r = int(sys.argv[2])
color = np.array(sys.argv[3:6], dtype=np.uint8)

im = Image.open(im_filename)
rgb = np.array(im.convert("RGB"), dtype=np.uint8) # RGB 0..255
rgb_ = rgb.astype(float)/255 # RGB 0..1

# Target (Dirichlet boundary conditions)
light = np.logical_and.reduce((rgb == color), axis=2, dtype=np.uint8)
ij2k = {}; k = 0 # Bijection: coordinates (i,j) <-> label k (unknowns)

nx, ny = light.shape
for i, j in np.ndindex(nx,ny):
    y, x = np.ogrid[-i:nx-i,-j:ny-j] # Axes of blur disk centered on each pixel
    disk = x*x + y*y <= r*r # Blur disk mask within range
    # >0 if pixel within range. =0 if not && Check if it's light or pixel to blur
    if light[disk].sum() and np.maximum(1-light[i][j], 0): ij2k[(i,j)] = (k:=k+1)

# Light follows Laplace's equation, discretized using finite
# difference method. Solve A*x=b in order find the unknowns.
```

```
A = np.empty(3, dtype=object) # Coefficient of the unknowns
b = np.zeros((k,3), dtype=float) # Right side (Dirichlet boundary conditions)
for i in range(3): A[i] = dok_matrix((k,k), dtype=float) # Sparse matrix

for (i,j), k in ij2k.items():

    # Five-point stencil
    ku = ij2k.get((i,j+1)) # up
    kd = ij2k.get((i,j-1)) # down
    kl = ij2k.get((i-1,j)) # left
    kr = ij2k.get((i+1,j)) # right

    # Three cases:
    # light=0 (no light -> no contribution)
    # light=1 (light -> Dirichlet)
    # i or j index out of range (neither -> no contribution)
    try: bu = light[i][j+1] * rgb_[i][j+1]
    except IndexError: bu = np.zeros(3, dtype=float)
    try: bd = light[i][j-1] * rgb_[i][j-1]
    except IndexError: bd = np.zeros(3, dtype=float)
    try: bl = light[i-1][j] * rgb_[i-1][j]
    except IndexError: bl = np.zeros(3, dtype=float)
    try: br = light[i+1][j] * rgb_[i+1][j]
    except IndexError: br = np.zeros(3, dtype=float)

    # k_ is None if it is not an unknown
    if ku: A[0][k-1,ku-1], A[1][k-1,ku-1], A[2][k-1,ku-1] = np.ones(3,dtype=float)
    if kd: A[0][k-1,kd-1], A[1][k-1,kd-1], A[2][k-1,kd-1] = np.ones(3,dtype=float)
    if kl: A[0][k-1,kl-1], A[1][k-1,kl-1], A[2][k-1,kl-1] = np.ones(3,dtype=float)
    if kr: A[0][k-1,kr-1], A[1][k-1,kr-1], A[2][k-1,kr-1] = np.ones(3,dtype=float)
    A[0][k-1, k-1], A[1][k-1, k-1], A[2][k-1, k-1] = -4*np.ones(3, dtype=float)
    b[k-1] = -bu-bd-bl-br

x = np.empty((A.shape[0], A[0].shape[0]))
for i in np.arange(A.shape[0]): x[i] = spsolve(A[i].tocsr(), b.transpose()[i])

# Add up light -> blur disks
for (i,j), k in ij2k.items(): rgb_[i][j] += x.T[k-1]
rgb = (np.clip(rgb_,0,1)*255.999).astype(np.uint8) RGB 0..1 to 0..255

fn, e = os.path.splitext(im_filename)
im_out = Image.fromarray(rgb)
if im.mode == "RGBA": im_out.putalpha(im.split()[-1])
im_out.save(fn+"_out"+e)
```