Tarea 2 - Dithering

Daniel Torres Robledo
shadow.cat6333@gmail.com

Andrés Fuentes Hernández
andres7233@hotmail.com

***Abstract*—En este documento se aborda el tema de *dithering*, mostrando la implementación de *dither ordenado*, *dither aleatorio* y *dither difusión de error*, para éste último en escala de grises y color.**

***Index Terms*—dithering, Floyd Steinberg, random, ordered, shared error, python, opencv**

I. Introducción

El uso del *dithering* permite poder mostrar imágenes en formatos que tengan una cantidad menor de color de profundidad, que para el caso de esta tarea, será 1 bit.

Esta técnica es empleada en las impresiones de periódicos o formatos en donde se tiene solo una tinta o color de profundidad de 1 bit, para lograr obtener una mayor resolución radiométrica, pero a su vez sacrificando resolución espacial, esto se logra aprovechando la integración espacial dentro del ojo humano [1].

El objetivo es mostrar las diferencias entre algunas implementaciones de *dithering*, que son: *dither ordenado*, *dither aleatorio* y *dither difusión de error*. Con variaciones en parámetros de algunas de éstas y una versión en color de difusión de error.

II. Implementación

Para ésta práctica se utilizó *python*, con las bibliotecas de *opencv, numpy, scipi y skimage*.

A continuación se describen características de cada uno de los métodos implementados y se muestran imágenes resultantes de aplicar dicho método.

A. Random Dithering

Esta variante de *dithering* es la más sencilla de implementar, sin embargo los resultados generalmente son los peores de las implementaciones. Se puede observar en la figura 1 la imagen original y la resultante.

Se puede observar que tanto en partes oscuras y claras hay puntos negros y blancos, por lo que esta técnica agrega mucho ruido al resultado.

A continuación se muestra el código de este método:

Fig. 1. Ejemplo de *random dither*.

```
for i in xrange(x):
        for j in xrange(y):
                if randint(0,255)<img[j,i]:
                        out[j,i] = 255
                else:
                        out[j,i] = 0
```

Fig. 2. Código para *random dither*.

B. Ordered Dithering

En esta técnica se utilizan matrices de $N \times N$ donde cada pixel de la imagen original es mapeado por la operación módulo N a la matriz definida, también es importante mencionar que el valor de cada pixel se escala de sus posibles valores en $[0, 255]$ a un intervalo de $[0, N]$ para poder comparar con los valores de la matriz, en donde si es menor o igual el resultado es 0, en otro caso es en 1.

Para este ejemplo, se utilizarán tres matrices, las cuales son [3]:

$$d_2 = \begin{matrix} 3 & 1 \\ 0 & 2 \end{matrix} \quad d_3 = \begin{matrix} 0 & 7 & 3 \\ 6 & 5 & 2 \\ 4 & 1 & 8 \end{matrix} \quad d_4 = \begin{matrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{matrix}$$

En la figura 3, se pueden observar los diferentes resultados de la misma imagen utilizando este método con diferentes tamaños de matrices, en orden de arriba a abajo d2, d3, d4.

Donde se puede observar que matrices más grandes generan mejores resultados. Pero una desventaja es que los patrones de

Fig. 3. Ejemplo de *ordered dither* utlizando las matrices d2, d3, d4.

las matrices son visibles, sobre todo en matrices pequeñas.

A continuación se muestra el código de este método:

```
for i in xrange(x):
        for j in xrange(y):
                if d[i%n][j%n]>(((n*n-0)*(img[j,i]-0))/(255-0))+0:
                        out[j,i] = 0
                else:
                        out[j,i] = 255
```

Fig. 4. Código para *random dither*.

C. Shared Error Dithering Color

El método de *Floyd Steinberg* para *dithering*, es un método de difusión del error y surge debido a que otros métodos de *dithering* como el ordenado dependen de crear una matriz $N \times N$ en la que se utilizan patrones de cuadros blancos y negros que emulan las escalas de grises, estos métodos tienen como consecuencia la perdida de resolución o bien una imagen de salida con dimensiones superiores a la original. Este método que surgió en el año de 1976, cuyo propósito es mantener el mismo tamaño de imagen repartiendo el error generado al binarizar ese pixel hacia los pixeles vecinos, pero solo aquellos *adelante* y por *debajo* de el es decir no se repartirá el error a pixeles ya visitados.

Para observar el comportamiento de este método se eligieron 3 imágenes, Lena, un bosque y una imagen artificial de triángulos y colores. En la figura 5 se puede observar que la imagen con *dithering* es una buena representación de la original, en donde los rasgos del rostro humano se pueden distinguir fácilmente, también es posible notar ciertos patrones en forma de *grecas* aparecen en la imagen esto puede deberse a la forma en la que el algoritmo distribuye el ruido.

También se utilizó una imagen de un bosque, ya que se quería ver si los detalles de las hojas de los pinos se preservaban en la imagen con *dithering*, además esta imagen tiene un brillo en la esquina superior izquierda.

En la figura 6 se puede ver que una gran porción de la imagen es blanca debido al brillo sin embargo los pinos y algunos de sus detalles aun se pueden distinguir con facilidad, incluso el paisaje de fondo ,el cual en la imagen original esta desenfocado y tiene un alto brillo, es posible distinguirlo en la imagen con *dithering*

Finalmente el algoritmo se puso a prueba con una imagen artificial es decir no es la representación de nuestro mundo. La última prueba consiste de una serie de triángulos de diferentes colores, el objetivo de esta prueba es ver la eficiencia del algoritmo con color uniformes y en los contornos. En la figura 7 se puede observar de una manera mas clara el ruido en forma de ”grecas” y en general aunque la imagen asemeja la original tiene un ruido significativo. Los triángulos mas afectados por este ruido son aquellos de color claro que se encuentran junto a un color mas obscuro. Es posible observar franjas del color adyacente en dichos triángulos.

Fig. 5. Ejemplo de *shared error dither* con Lena.

Fig. 6. Ejemplo de *shared error dither* con el bosque.

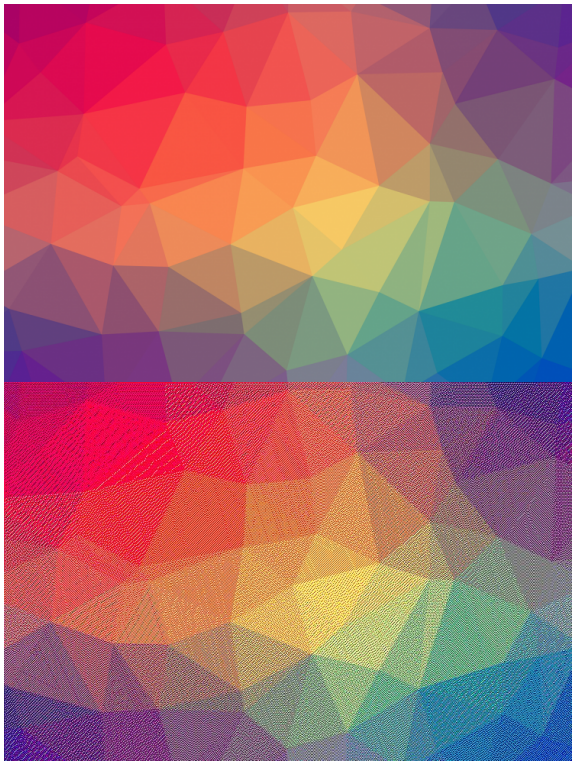

Fig. 7. Ejemplo de *shared error dither* con triángulos.

D. Shared Error Dithering

Esta implementación, al igual que la anterior, pero en un solo canal y un bit de profundidad, se puede apreciar un resultado con calidad similar al de *Ordered Dither* de una matriz de 4×4, sin embargo, aquí no hay patrones visibles.

Sin embargo, este algoritmo es el que tarda más en ejecutarse pero ofrece resultados buenos, manteniendo características de la imagen, como los bordes del sombrero y partes de las plumas del mismo. A continuación se muestra el código de este método:

III. Conclusiones

En general, los diferentes métodos de *dithering* implementados muestran resultados donde se puede apreciar la imagen original con al menos unos detalles.

El peor de los métodos fue el aleatorio, donde la imagen resultante tiene mucho ruido.

Posteriormente, el *dither* ordenado muestra un buen desempeño, pero depende de poder crear las matrices de $N \times N$, mientras más grande sea N, los resultados son mejores, sin embargo, los patrones de estas matrices son claros a simple vista.

El algoritmo de *Floyd Steinberg* para *dithering* produce una buena aproximación a la imagen original utilizando solo 1 bit por color. Aunque aun es posible ver ciertos patrones sobre todo en lugares de color uniforme que tienen vecindad con un color mas obscuro, cierta *contaminación* se puede producir.

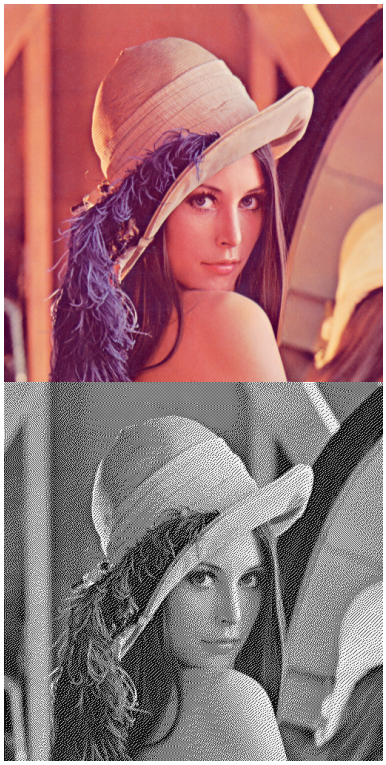

Fig. 8. Ejemplo de *shared error dither* en 1 bit.

```
for i in xrange(x):
        for j in xrange(y):
                closest = 255 if img[j,i]>(255.0/2) else 0
                error = img[j,i]-closest
                img[j,i] = closest
                if i<x-1:
                        img[j,i+1] += (7.0*error)/16.0
                if i>0 and j<y-1:
                        img[j+1,i-1] += (3.0*error)/16.0
                if j<y-1:
                        img[j+1,i] += (5.0*error)/16.0
                if i<x-1 and j<y-1:
                        img[j+1,i+1] += (1.0*error)/16.0
```

Fig. 9. Código para *random dither*.

Este último algoritmo muestra los mejores resultados, preservando suficientes detalles de la imagen original, sin embargo el tiempo de procesamiento es el mayor de todos.

Por lo que se recomienda *Shared Error Dithering* para generar este tipo de imágenes cuando el dispositivo en donde se visualizará tiene solo un color, sin embargo si lo que se busca es velocidad de respuesta *Ordered Dithering* es una buena opción, con una matriz de tamaño 3 o 4.

REFERENCES

[1] Team Veryovka-Buchanan, *Developed Algorithms*. Avalible at *https://www.visgraf.impa.br/Courses/ip00/proj/Dithering1/algoritmos_desenvolvidos.htm*. [Accessed Agosto 25, 2019]

[2] Tanner Helland, *Image Dithering: Eleven Algorithms and Source Code*. Avalible at *http://www.tannerhelland.com/4660/dithering-eleven-algorithms-source-code*. [Accessed Agosto 25, 2019]

[3] Ordered Dithering. [Online]. Avalible at *https://en.wikipedia.org/wiki/Ordered_dithering*. [Accessed Agosto 25, 2019]