

Informe de Laboratorio 04

Octree

Integrantes:

Ruelas Quenaya Merisabel
Villarroel Ramos Fiorela Estefany

Laboratorio de Estructuras de Datos Avanzados

Grupo A



Profesor: Rolando Cardenas Talavera

16 de noviembre de 2022

Laboratorio 04

1. Competencia del Curso

Conoce e investiga los métodos de acceso multidimensional, métrico y aproximado

2. Competencia del Laboratorio

- ✓ Describir, implementar la estructura de datos OcTree
- ✓ Analizar el funcionamiento de la estructura OcTree e image quantization.

3. Equipos y Materiales

- ✓ Un computador, se formaran grupos de trabajo no mayor a 3 personas
- ✓ Lenguaje de Programación (C++, Python, java, c, u otros)

4. Desarrollo

4.1. Actividad 1

Implemente el algoritmo de Color Quantization utilizando un Octree en C++ u otro lenguaje de programación seleccionado.

4.1.1. Algoritmo

Octtree es un árbol donde cada nodo tiene hasta 8 hijos. El nodo hoja no tiene hijos activos. Cada nodo de hoja tiene una cantidad de píxeles con este color y valor de color.

Adición de un nuevo color al octárbol

Comienza en el nivel 0. Por ejemplo, un píxel de color RGB es (90, 13, 157). En binario es (01011010, 01110001, 10011101). El índice del nodo del siguiente nivel se calcula de la siguiente manera: Escribir en binario los bits R, G y B, empezando por el MSB, para el nivel actual. Así, el índice será de 000 a 111 (binario), es decir, de 0 a 7 (decimal). Si la profundidad máxima del árbol es inferior a 8, sólo importarán los primeros bits del color.

Reducción

Para hacer la paleta de colores de la imagen con, por ejemplo, 256 colores como máximo, a partir de la paleta con muchos más colores hay que reducir las hojas del árbol. La reducción de los nodos: Como tenemos una suma de los valores R, G y B y el número de píxeles con este color, podemos añadir el recuento de todos los píxeles de las hojas y los canales de color al nodo padre y convertirlo en un nodo hoja (ni siquiera podríamos eliminarlo, porque el método .ºbtener hojas”no profundizará si el nodo actual es una

hoja). La reducción continúa mientras el recuento de hojas es superior al máximo de colores necesario (en nuestro caso 256). La principal desventaja de este enfoque es que se pueden reducir hasta 8 hojas del nodo y la paleta podría tener sólo 248 colores (en el peor de los casos) en lugar de los 256 colores esperados. En cuanto el número de hojas sea inferior o igual al máximo de colores necesarios, podremos construir una paleta.

Construcción de la paleta

La paleta se rellena con los colores medios de cada hoja. Como cada hoja tiene el número de píxeles con color y la suma de los valores R, G y B del color, el color medio se puede obtener dividiendo los canales de color por el número de píxeles: `palette_color = (color.R / pixel_count, color.G / pixel_count, color.B / pixel_count)`.

4.2. Actividad 2

Implementar un algoritmo utilizando un octree para reducir la cantidad de colores de dicha imagen, por ejemplo en la Fig. 5 y Fig. 6, la cantidad de colores se ha reducido a 256 y 64 respectivamente, además también podemos ver la paleta de colores utilizada. Su algoritmo debe retornar tanto la imagen reducida como la paleta de colores.

```
1 from PIL import Image
2
3 class Color(object): # Clase Color
4     def __init__(self, red=0, green=0, blue=0): # Constructor
5         self.red = red
6         self.green = green
7         self.blue = blue
8
9 class OctreeNode(object): # Clase Nodo del Octree
10    def __init__(self, level, parent): # Constructor
11        self.color = Color(0, 0, 0)
12        self.cntPixel = 0
13        self.indPaleta = 0
14        self.children = [None, None, None, None, None, None, None, None]
15        # Ocho hijos del octree
16        if level < OctreeQuantizer.MAX_DEPTH - 1: # añade un nodo al
17            # nivel actual
18            parent.AddNodeByLevel(level, self)
19    def getNodesLeaf(self): # Obtener todos los nodos hoja
20        nodesLeaf = []
21        for i in range(8):
22            if self.children[i]:
23                if self.cntPixel > 0:
24                    nodesLeaf.append(self.children[i])
```

```

23         else:
24             nodesLeaf.extend(self.children[i].getNodesLeaf())
25     return nodesLeaf
26 def getCntPixelNodes(self): # Obtener una suma de la cantidad de
    ↪ píxeles para el nodo y sus hijos
27     sum = self.cntPixel
28     for i in range(8):
29         if self.children[i]:
30             sum += self.children[i].cntPixel
31     return sum
32 def addColor(self, color, level, parent): # Añadir un color al
    ↪ árbol
33     if level >= OctreeQuantizer.MAX_DEPTH:
34         self.color.red += color.red
35         self.color.green += color.green
36         self.color.blue += color.blue
37         self.cntPixel += 1
38     return
39     index = self.getColorLevel(color, level)
40     if not self.children[index]:
41         self.children[index] = OctreeNode(level, parent)
42     self.children[index].addColor(color, level + 1, parent)
43 def getIndPaleta(self, color, level): # Obtiene el índice de
    ↪ paleta para color. Utiliza nivel para ir un nivel más allá si
    ↪ el nodo no es una hoja
44     if self.cntPixel > 0:
45         return self.indPaleta
46     index = self.getColorLevel(color, level)
47     if self.children[index]:
48         return self.children[index].getIndPaleta(color, level + 1)
49     else:
50         for i in range(8): # Obtener el índice de paleta para el
            ↪ primer nodo hijo encontrado
51             if self.children[i]:
52                 return self.children[i].getIndPaleta(color, level +
                    ↪ 1)
53 def deleteLeaves(self): # Añade el recuento de píxeles y los
    ↪ canales de color de todos los hijos al nodo padre. Devuelve el
    ↪ número de hojas eliminadas
54     ans = 0
55     for i in range(8):
56         node = self.children[i]
57         if node:

```

```

58         self.color.red += node.color.red
59         self.color.green += node.color.green
60         self.color.blue += node.color.blue
61         self.cntPixel += node.cntPixel
62         ans += 1
63     return ans - 1
64 def getColorLevel(self, color, level): # Obtener el índice de color
    ↪ para el siguiente nivel
65     index = 0
66     mask = 0x80 >> level
67     if color.red & mask:
68         index |= 4
69     if color.green & mask:
70         index |= 2
71     if color.blue & mask:
72         index |= 1
73     return index
74 def getColor(self): # Obtener el color medio
75     return Color(
76         self.color.red / self.cntPixel,
77         self.color.green / self.cntPixel,
78         self.color.blue / self.cntPixel)
79
80 class OctreeQuantizer(object): # Clase Octree Quantizer
81     MAX_DEPTH = 8 # Para limitar el número de niveles
82     def __init__(self): # Constructor
83         self.levels = {i: [] for i in range(OctreeQuantizer.MAX_DEPTH)}
84         self.root = OctreeNode(0, self)
85     def getLeaves(self): # Obtener todas las hojas
86         return [node for node in self.root.getNodesLeaf()]
87     def AddNodeByLevel(self, level, node): # Añadir nodo a los nodos en
    ↪ nivel
88         self.levels[level].append(node)
89     def addColor(self, color): # Añadir color al octree
90         self.root.addColor(color, 0, self) # pasa el valor de self como
    ↪ `parent` para guardar los nodos en los niveles dict
91     def constructPaleta(self, color_count):
92         palette = []
93         indPaleta = 0
94         leaf_count = len(self.getLeaves())
95         # Reduce nodos. Se pueden reducir hasta 8 hojas y la paleta
    ↪ tendrá solo 248 colores (en el peor de los casos) en lugar
    ↪ de los 256 colores esperados

```

```

96     for level in range(OctreeQuantizer.MAX_DEPTH - 1, -1, -1):
97         if self.levels[level]:
98             for node in self.levels[level]:
99                 leaf_count -= node.deleteLeaves()
100                 if leaf_count <= color_count:
101                     break
102                 if leaf_count <= color_count:
103                     break
104                 self.levels[level] = []
105     for node in self.getLeaves(): # Construir la paleta
106         if indPaleta >= color_count:
107             break
108         if self.cntPixel > 0:
109             palette.append(node.getColor())
110             node.indPaleta = indPaleta
111             indPaleta += 1
112     return palette
113 def getIndPaleta(self, color): # Obtener el índice de la paleta
114     ↪ para color
115     return self.root.getIndPaleta(color, 0)
116
117 def main():
118     image = Image.open('rainbow.png')
119     pixels = image.load()
120     width, height = image.size
121     octree = OctreeQuantizer() # Inicializando el octree
122     # Añadir los colores al octree
123     for j in range(height):
124         for i in range(width):
125             octree.addColor(Color(*pixels[i, j]))
126     # 256 colores para una imagen de salida de 8 bits por pixel
127     palette = octree.constructPaleta(256)
128     # Crear paleta para 256 colores y guardar la paleta como archivo
129     palette_image = Image.new('RGB', (16, 16))
130     palette_pixels = palette_image.load()
131     for i, color in enumerate(palette):
132         palette_pixels[i % 16, i / 16] = (int(color.red), int
133             ↪ (color.green), int (color.blue))
134     palette_image.save('rainbow_palette.png')
135     # Guardar la imagen resultante
136     out_image = Image.new('RGB', (width, height))
137     out_pixels = out_image.load()
138     for j in range(height):

```

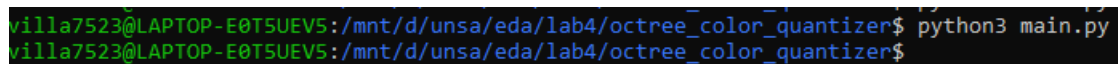
```

137     for i in range(width):
138         index = octree.getIndPaleta(Color(*pixels[i, j]))
139         color = palette[index]
140         out_pixels[i, j] = (int(color.red), int(color.green),
141                             ↪ int(color.blue))
142
141     out_image.save('rainbow_out.png')
142
143 if __name__ == '__main__':
144     main()

```

4.3. Resultados

4.3.1. Ejecución del código

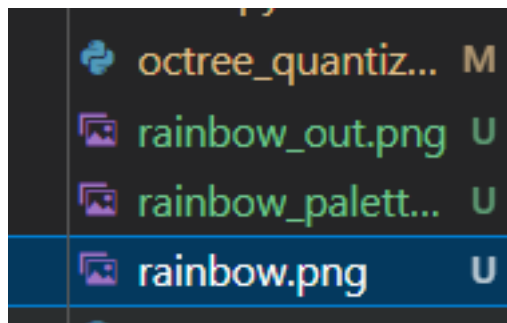


```

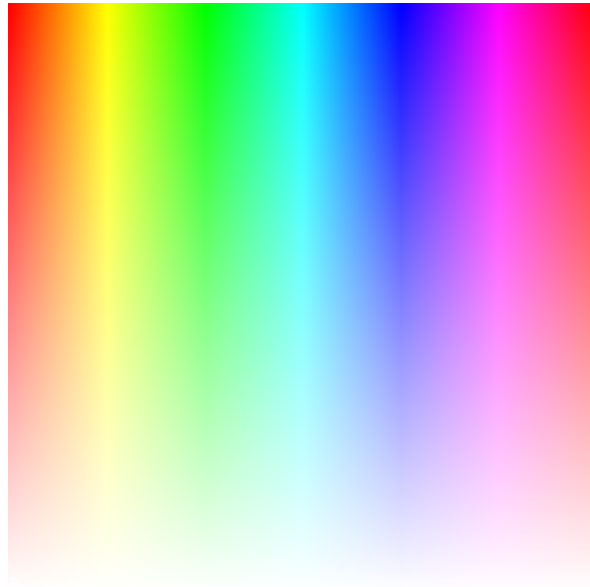
villa7523@LAPTOP-E0T5UEV5:/mnt/d/uns/eda/lab4/octree_color_quantizer$ python3 main.py
villa7523@LAPTOP-E0T5UEV5:/mnt/d/uns/eda/lab4/octree_color_quantizer$

```

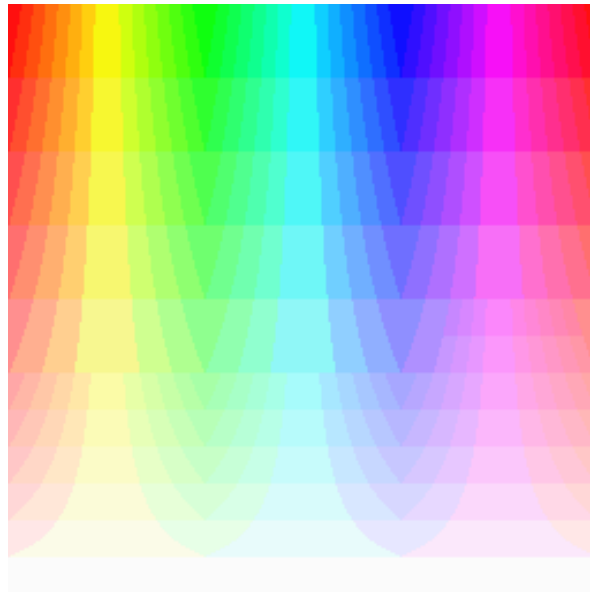
4.3.2. Archivos generados



4.3.3. Imágen original (24 bits):



4.3.4. Imágen reducida a (8 bits):



4.3.5. Imágen reducida a (8 bits):

