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

20 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.

```python
from PIL import Image # Libreria que agrega soporte para abrir, manipular y guardar muchos formatos de archivo de imagen diferentes
import argparse

class Color(object): # Clase Color: Almacena atributos RGB de un color
    def __init__(self, red=0, green=0, blue=0): # Constructor
        self.red = red
        self.green = green
        self.blue = blue

class NodeOctree(object): # Clase Nodo del Octree
    def __init__(self, level, parent): # Constructor
        self.color = Color(0, 0, 0)
        self.cntPixel = 0
        self.indPaleta = 0
        self.children = [None for _ in range(8)] # Ocho hijos del octree
        if level < Octree.MAX_DEPTH - 1: # añade un nodo al nivel actual
            parent.AddNodeByLevel(level, self)
    def getNodesLeaf(self): # Obtener todos los nodos hoja
        nodesLeaf = []
        for i in range(8):
```

```
21              if self.children[i]:
22                  if self.children[i].cntPixel > 0:
23                      nodesLeaf.append(self.children[i])
24                  else:
25                      nodesLeaf.extend(self.children[i].getNodesLeaf())
26          return nodesLeaf
27      def getCntPixelNodes(self): # Obtener una suma de la cantidad de
        ↪  píxeles para el nodo y sus hijos
28          sum = self.cntPixel
29          for i in range(8):
30              if self.children[i]:
31                  sum += self.children[i].cntPixel
32          return sum
33      def addColor(self, color, level, parent): # Añadir un color al
        ↪  arbol
34          if level >= Octree.MAX_DEPTH:
35              self.color.red += color.red
36              self.color.green += color.green
37              self.color.blue += color.blue
38              self.cntPixel += 1
39              return
40          index = self.getColorLevel(color, level)
41          if not self.children[index]:
42              self.children[index] = NodeOctree(level, parent)
43          self.children[index].addColor(color, level + 1, parent)
44      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
45          if self.cntPixel > 0:
46              return self.indPaleta
47          index = self.getColorLevel(color, level)
48          if self.children[index]:
49              return self.children[index].getIndPaleta(color, level + 1)
50          else:
51              for i in range(8): # Obtener el índice de paleta para el
                ↪  primer nodo hijo encontrado
52                  if self.children[i]:
53                      return self.children[i].getIndPaleta(color, level +
                        ↪  1)
54      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
55          ans = 0
```

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

```
96          # 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
97          for level in range(Octree.MAX_DEPTH - 1, -1, -1):
98              if self.levels[level]:
99                  for node in self.levels[level]:
100                     leaf_count -= node.deleteLeaves()
101                     if leaf_count <= color_count:
102                         break
103                 if leaf_count <= color_count:
104                     break
105                 self.levels[level] = []
106         for node in self.getLeaves(): # Construir la paleta
107             if indPaleta >= color_count:
108                 break
109             if node.cntPixel > 0:
110                 palette.append(node.getColor())
111             node.indPaleta = indPaleta
112             indPaleta += 1
113         return palette
114     def getIndPaleta(self, color): #  Obtener el índice de la paleta
        ↪ para color
115         return self.root.getIndPaleta(color, 0)
116
117 def main():
118     parser = argparse.ArgumentParser(description="Octree_Quantizer")
        ↪ #establecer algoritmos
119     parser.add_argument('input_file')
120
121     args = vars(parser.parse_args())
122     print(args)
123     image = Image.open(args['input_file'])
124     pixels = image.load()
125     width, height = image.size
126     octree = Octree() # Inicializando el octree
127     octree_64 = Octree() # Inicializando el octree
128
129     # Añadir los colores al octree
130     for j in range(height):
131         for i in range(width):
132             color = Color(*pixels[i, j])
133             octree.addColor(color)
134             octree_64.addColor(color)
```

```
135
136     ##################################### 256-bits
        ↪  ########################################33
137     # 256 colores para una imagen de salida de 8 bits por pixel
138     palette_256 = octree.constructPaleta(256)
139
140     # Crear paleta para 256 colores y guardar la paleta como archivo
141     palette_256_image = Image.new('RGB', (16, 16))
142     palette_256_pixels = palette_256_image.load()
143     for i, color in enumerate(palette_256):
144         palette_256_pixels[i % 16, i / 16] = (int(color.red), int
            ↪  (color.green), int (color.blue))
145
146     #Estableciendo el nombre del archivo de salida(paleta)
147     name = args['input_file'].split('.')
148     name = name[0] + '_256_palette.' + name[1]
149     palette_256_image.save('Img/' + name)
150
151     # Guardar la imagen resultante
152     out_image = Image.new('RGB', (width, height))
153     out_pixels = out_image.load()
154     for j in range(height):
155         for i in range(width):
156             index = octree.getIndPaleta(Color(*pixels[i, j]))
157             color = palette_256[index]
158             out_pixels[i, j] = (int(color.red), int(color.green),
                ↪  int(color.blue))
159     #Estableciendo el nombre del archivo de salida(paleta)
160     name = args['input_file'].split('.')
161     name = name[0] + '_256_ImageReduced.' + name[1]
162     out_image.save('Img/' + name)
163
164     ##################################### 64-bits
        ↪  ########################################33
165     # 64 colores para una imagen de salida de 6 bits por pixel
166     palette_64 = octree_64.constructPaleta(64)
167
168     # Crear paleta para 64 colores y guardar la paleta como archivo
169     palette_64_image = Image.new('RGB', (8, 8))
170     palette_64_pixels = palette_64_image.load()
171     for i, color in enumerate(palette_64):
172         palette_64_pixels[i % 8, i / 8] = (int(color.red), int
            ↪  (color.green), int (color.blue))
```

```
173
174     #Estableciendo el nombre del archivo de salida(paleta)
175     name = args['input_file'].split('.')
176     name = name[0] + '_64_palette.' + name[1]
177     palette_64_image.save('Img/' + name)
178
179     # Guardar la imagen resultante
180     out_image_64 = Image.new('RGB', (width, height))
181     out_pixels_64 = out_image_64.load()
182     for j in range(height):
183         for i in range(width):
184             index_64 = octree_64.getIndPaleta(Color(*pixels[i, j]))
185             color = palette_64[index_64]
186             out_pixels_64[i, j] = (int(color.red), int(color.green),
                ↪  int(color.blue))
187     #Estableciendo el nombre del archivo de salida(paleta)
188     name = args['input_file'].split('.')
189     name = name[0] + '_64_ImageReduced.' + name[1]
190     out_image_64.save('Img/' + name)
191
192 if __name__ == '__main__':
193     main()
```

4.3. Resultados

Figura 1. Ejecución del programa

```
villa7523@LAPTOP-E0T5UEV5:/mnt/d/unsa/eda/q/EDALABORATORIO4$ python3 code.py paisaje.jpg
{'input_file': 'paisaje.jpg'}
villa7523@LAPTOP-E0T5UEV5:/mnt/d/unsa/eda/q/EDALABORATORIO4$ python3 code.py flor.jpg
{'input_file': 'flor.jpg'}
villa7523@LAPTOP-E0T5UEV5:/mnt/d/unsa/eda/q/EDALABORATORIO4$ python3 code.py rainbow.png
{'input_file': 'rainbow.png'}
villa7523@LAPTOP-E0T5UEV5:/mnt/d/unsa/eda/q/EDALABORATORIO4$
```

4.3.1. Imagen 1 *(Rainbow.png)*

Figura 2. Imagen original (24 bits)

Figura 3. Imágen con reducción (**256 colores**)

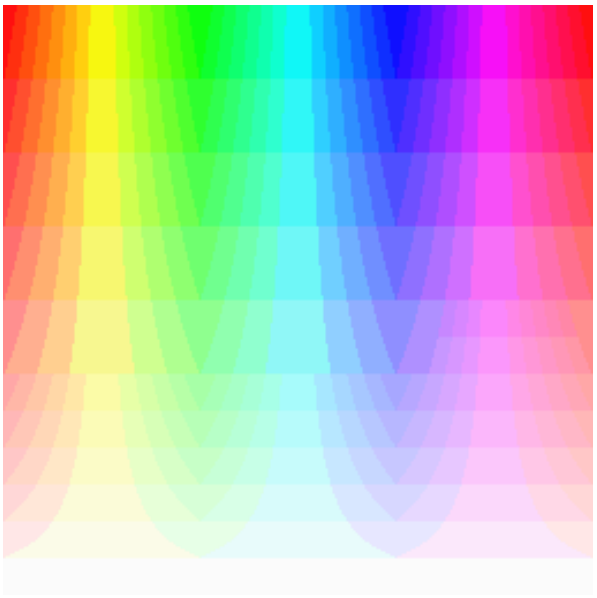


Figura 4. Paleta de tonalidades usadas (**256 colores**

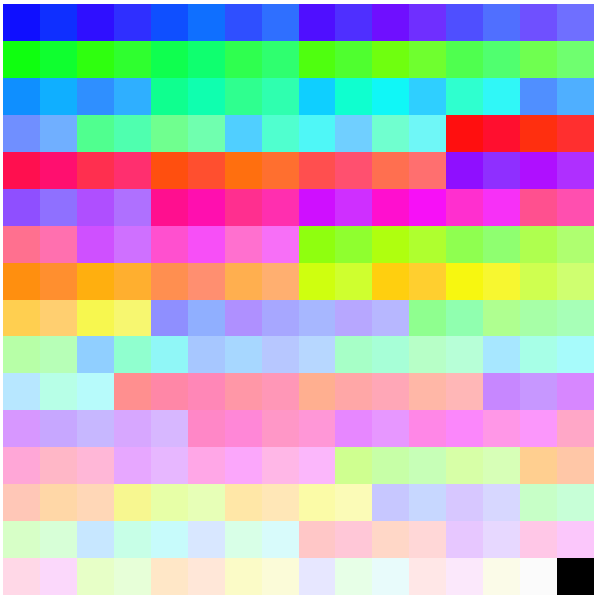


Figura 5. Imágen con reducción (**64 colores**)

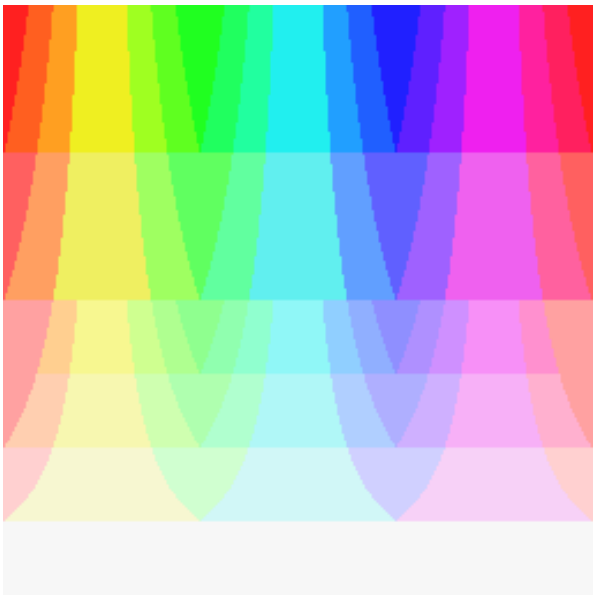
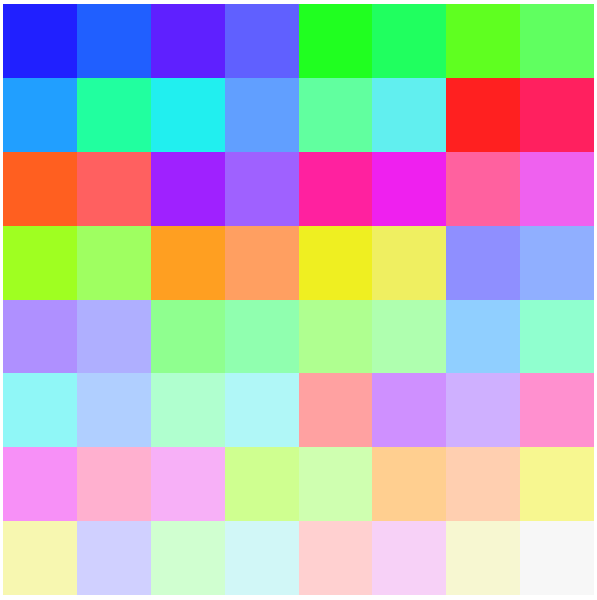


Figura 6. Paleta de tonalidades usadas (**64 colores**



4.3.2. Imagen 2 *(Flor.jpg)*

Figura 7. Imagen original (24 bits)

Figura 8. Imágen con reducción (**256 colores**)

Figura 9. Paleta de tonalidades usadas (**256 colores**

Figura 10. Imágen con reducción (**64 colores**)

Figura 11. Paleta de tonalidades usadas (**64 colores**

4.3.3. Imagen 3 *(paisaje.jpg)*

Figura 12. Imagen original (24 bits)

Figura 13. Imágen con reducción (**256 colores**)

Figura 14. Paleta de tonalidades usadas (**256 colores**

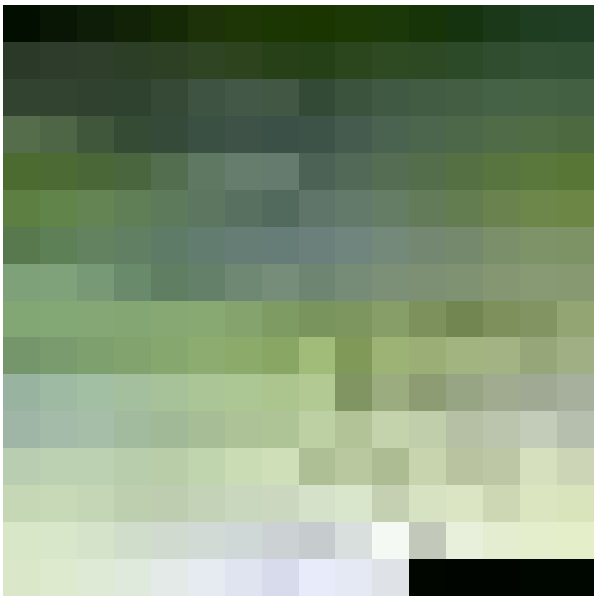


Figura 15. Imágen con reducción (**64 colores**)

Figura 16. Paleta de tonalidades usadas (**64 colores**