Universidad de Nacional de San Agustín
Facultad de Producción y Servicios
Escuela de Ciencia de la Computación

Practica 4

Octree Color Quantization

Integrantes: Mita Yagua Lesly Yaneth
Pinto Medina Brian
Ccari Quispe José
Gonza Condori, Gabriel
Flores Herrera, Jefferson
Profesor: MSc. Vicente Machaca Arceda

Fecha de entrega: 9 de noviembre de 2020
Arequipa, Perú

Índice de Contenidos

1. Introducción **1**
1.1. Presentación y objetivos 1
1.2. Herramientas 1
1.3. Estructura del documento 1

2. Implementación **2**
2.1. Clase node del Octree 2
2.2. Método Fill 3
2.3. Método Reduction 4
2.4. Método Reconstruction 4
2.5. Método Pallete 5

3. Evaluación y Pruebas **6**

4. Conclusión **10**

Anexo A. Código fuente **11**

Anexo B. Capturas de pantalla **17**

Anexo C. Repositorios **20**

Índice de Figuras

1. Herramientas utilizadas 1
2. Visualización 1 6
3. Visualización 2 6
4. Prueba 1 7
5. Prueba 2 8
6. Prueba 3 9
B.1. Visualización 1 17
B.2. Visualización 2 18
B.3. Captura de Pantalla de main.html 18
B.4. Captura de Pantalla de loadImg.js 19
B.5. Captura de Pantalla de octree.js 19

Índice de Códigos

1. Nodo Octree 2
2. Método Fill 3
3. Método Reduction 4
4. octree.js 4
5. octree.js 5

A.1. main.html 11
A.2. octree.js 11
A.3. loadImg.js 14

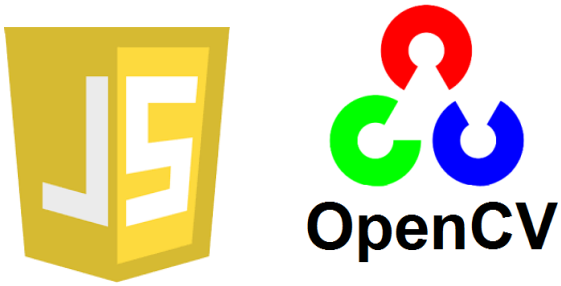
1. Introducción

1.1. Presentación y objetivos

La estructura de datos multidimensional Octree implementada anteriormente el cual posee distintas aplicaciones siendo una de ellas el procesamiento de imágenes. Por ello se realizo la implementación de Color-Quantization que es un algoritmo fascinante y sorprendentemente simple que nos permite reducir el número de colores únicos en una imagen manteniendo el aspecto general de la imagen. En este presente trabajo buscaremos adaptar la estructura multidimensional Octree en función al algoritmo Color-Quantization.

1.2. Herramientas

Para la implementación como lenguaje de programación se utilizó Javascript para la implementación de la estructura y para el procesamiento de la imagen se utilizó Opencv , esto debido a que es muy práctico ya que simplifica en gran medida el proceso de tratamiento de imágenes .



(a) Javascript (b) Opencv

Figura 1: Herramientas utilizadas

1.3. Estructura del documento

El presente documento está dividido en una serie de secciones que conforman el proceso de desarrollo del proyecto, detalladas a continuación:

- **Implementación:** Se detalla el todo el desarrollo de la implementación de todos los métodos necesarios para Color-Quantization a partir de la estructura multidimensional OcTree.
- **Evaluación y Pruebas:** Se detalla los resultados de pruebas en Color-Quantization para la comprobación del correcto funcionamiento mediante una serie de pruebas.
- **Conclusión:** Se detallan las observaciones sobre Color-Quantization, en función a la implementación y capacidades de este.

2. Implementación

Para la implementación de Color-Quantization adaptamos tanto sus métodos como estructura de Octree de tal forma que se adecue a Color-Quantization que es un algoritmo fascinante y sorprendentemente simple que nos permite reducir el número de colores únicos en una imagen manteniendo el aspecto general de la imagen.
Los diferentes métodos tanto en la clase nodo como en la clase octree se detallaran a continuación en cada una se las secciones.

2.1. Clase node del Octree

Se definió la clase Node de la estructural multidimensional OcTree con los siguientes siguientes funciones

- **isLeaf :** Determina un flag para saber si es un nodo hoja.
- **setLeaf :** Define un nodo como nodo hoja.
- **insertColor :** Inserta un color RGB dentro del nodo.
- **setColor :** Establece un color RGB dentro del nodo.
- **initHeight :** Inicializa la altura del OcTree

Código 1: Nodo Octree

```
class Node{
    constructor(leaf,color){
        this.leaf=leaf;
        this.color=color;
        this.pixelCount= 0;
        this.colorInRgb;
        this.array= new ArrayNodes;
        this.#isLeaf();
    }

    #isLeaf(){
        if(!this.leaf)
            this.array.init();
    }

    setLeaf(leaf){
        this.leaf = leaf;
        this.#isLeaf();
    }

    insertColor(colorRgb, colorInOctal, indice){
        if(this.leaf){
            this.pixelCount ++;
            this.colorInRgb = colorRgb;
        }
```

```
26          if(indice == 8)
27              return;
28          this.array.array[colorInOctal[indice]].color = colorInOctal[indice];
29          this.array.array[colorInOctal[indice]].insertColor(colorRgb, colorInOctal, indice + 1);
30      }
31
32      setColor(color, colorInOctal, indice){
33          if(this.leaf){
34              color.red = this.colorInRgb.red;
35              color.green = this.colorInRgb.green;
36              color.blue = this.colorInRgb.blue;
37              return;
38          }
39          this.array.array[colorInOctal[indice]].color = colorInOctal[indice];
40          this.array.array[colorInOctal[indice]].setColor(color, colorInOctal, indice + 1);
41      }
42
43      initHeight(height){
44          if(height==1){
45              for(let i = 0; i < 8; ++i)
46                  this.array.array[i].setLeaf(true);
47              return;
48          }
49          for(var i=0;i<8;i++){
50              this.array.array[i].setLeaf(false);
51              this.array.array[i].initHeight(height-1);
52          }
53      }
54  }
```

2.2. Método Fill

Se definió este método con el propósito de que lea los píxeles de una imagen y construya el Oc-Tree en función al algoritmo Color-Quantization

Código 2: Método Fill

```
1  fill (matriz){
2      for(let i = 0; i < matriz.length; ++i){
3          let red = numberToBinaryString(matriz[i].red);
4          let green = numberToBinaryString(matriz[i].green);
5          let blue = numberToBinaryString(matriz[i].blue);
6          let colorFinalTransformation = [];
7          for(let j = 0; j< 8; ++j){
8              let octal = red[j] + green[j] + blue[j];
9              let octalInDecimal = parseInt(octal, 2);
10             colorFinalTransformation.push(octalInDecimal);
11         }
12         this.head.insertColor(matriz[i], colorFinalTransformation, 0);
13     }
14 }
```

2.3. Método Reduction

Se definió el método redcution con el proposito de eliminar el ultimo nivel de la estructura Oc-Tree y acumular todos los valores RGB al nodo padre.

Código 3: Método Reduction

```
reduction(node){
    if(node.array.array[0].leaf == true ) {
        var red = 0, green = 0, blue = 0, pixelCount = 0;

        for(let i = 0; i < 8; ++i) {
            if(node.array.array[i].pixelCount === 0) {
                continue;
            }
            red += node.array.array[i].colorInRgb.red * node.array.array[i].pixelCount;
            green += node.array.array[i].colorInRgb.green * node.array.array[i].pixelCount;
            blue += node.array.array[i].colorInRgb.blue * node.array.array[i].pixelCount;
            pixelCount += node.array.array[i].pixelCount;
        }

        node.leaf = true;
        node.pixelCount = pixelCount;
        if(pixelCount > 0 ) {
            var color = new Color( parseInt(red / pixelCount),parseInt(green / pixelCount),
    ↪ parseInt(blue / pixelCount ));
            node.colorInRgb = color;
        }
        node.array = [];

    } else {
        for(let i = 0; i < 8; ++i) {
            this.reduction(node.array.array[i]);
        }
    }
}
```

2.4. Método Reconstruction

Este método tiene como parámetro una matriz que contiene todos los colores RGB de la imagen ya reducida , así pues asignamos a cada pixel su nuevo color.

Código 4: octree.js

```
reconstruction(matriz){
        for(let i = 0; i < matriz.length; ++i){
            let red = numberToBinaryString(matriz[i].red);
            let green = numberToBinaryString(matriz[i].green);
            let blue = numberToBinaryString(matriz[i].blue);
            let colorFinalTransformation = [];
            for(let j = 0; j< 8; ++j){
                let octal = red[j] + green[j] + blue[j];
```

```
                let octalInDecimal = parseInt(octal, 2);
                colorFinalTransformation.push(octalInDecimal);
            }
            this.head.setColor(matriz[i], colorFinalTransformation, 0);

        }
    }
```

2.5. Método Pallete

Este método tiene como parámetros el nodo raiz del Octree y una matriz vacia para almacenar los colores RGB. Lo que hacemos es recorrer todo el Octree hasta las hojas y recolectar todos los colores almacenados.

Código 5: octree.js

```
  pallete(nodo, mat){
        if(nodo.leaf && nodo.pixelCount > 0){
            mat.push(nodo.colorInRgb);
            return;
        }else if(nodo.leaf && nodo.color === null){
            return;
        }
        for(let i =0; i < 8; ++i){
            this.pallete(nodo.array.array[i], mat);
        }
    }
```

3. Evaluación y Pruebas

La visualización de Color-Quantization en el navegador web fue lo siguiente

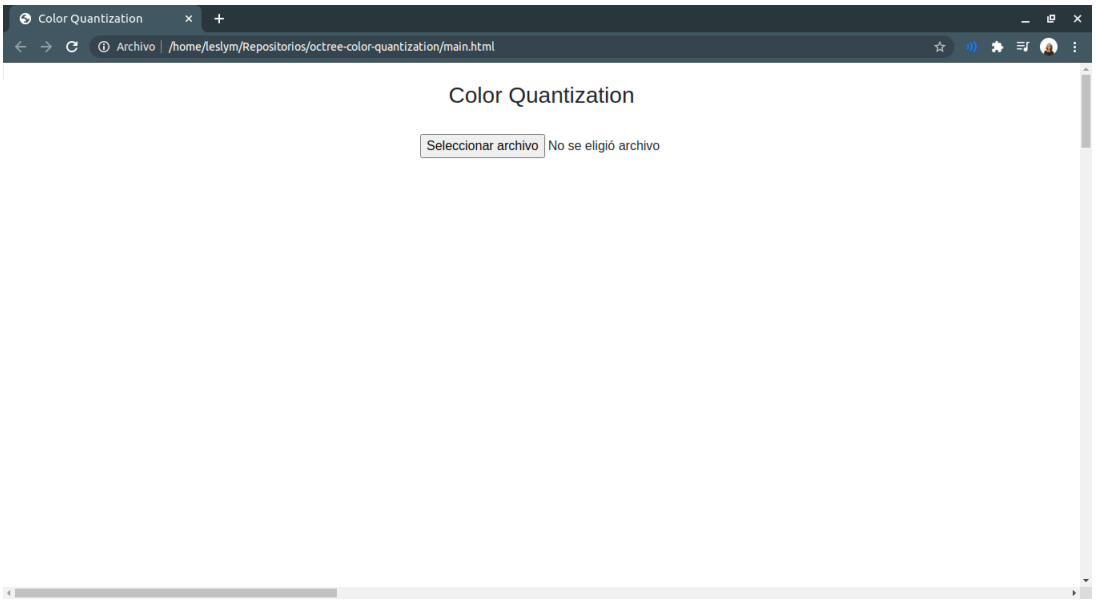


Figura 2: Visualización 1

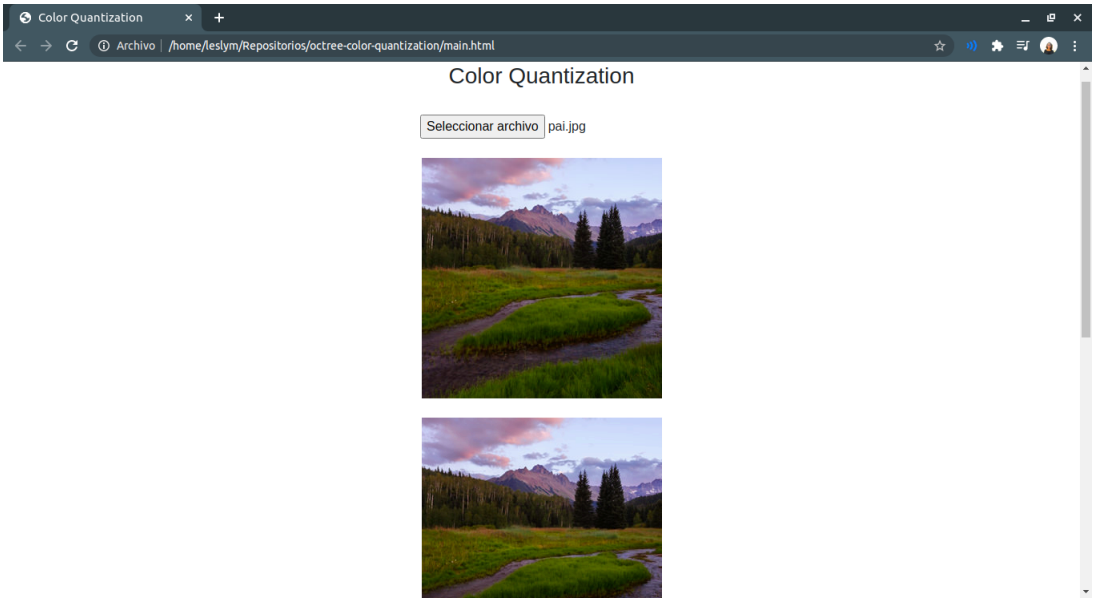
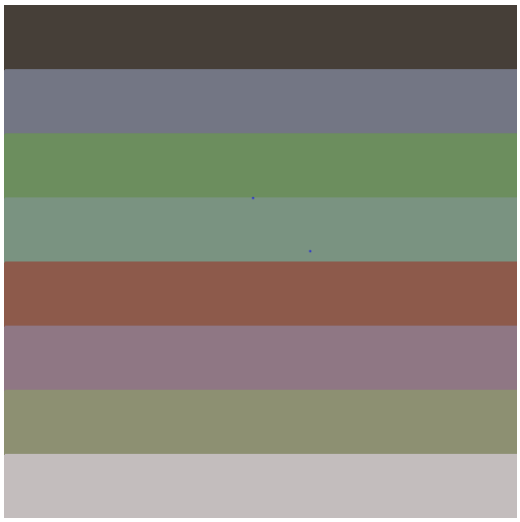


Figura 3: Visualización 2

Aplicaremos Color-Quantization a diversas imágenes y veremos cual es el resultado:

(a) Imagen original

(b) Imagen reducida en 7

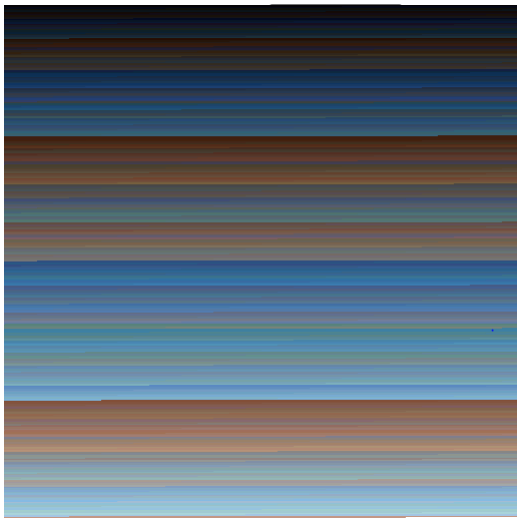


(c) pallet

Figura 4: Prueba 1

(a) Imagen original

(b) Imagen reducida en 4



(c) pallet

Figura 5: Prueba 2

(a) Imagen original

(b) Imagen reducida en 2

(c) pallet

Figura 6: Prueba 3

4. Conclusión

En conclusión podemos decir que Color-Quantization haciendo uso la estructura de datos Octree es un aplicación sorprendentemente simple que si bien es cierto no demasiado óptima (ya que hay otros métodos mucho más eficaces) este cumple muy bien con su objetivo y de una manera sencilla.
Algunos editores de gráficos vectoriales también utilizan Color-Quantization, especialmente para técnicas de trama a vector que crean trazados de imágenes de mapa de bits con la ayuda de la detección de bordes .

Anexo A. Código fuente

Código A.1: main.html

```
<!DOCTYPE html>
<html>
    <head>
        <script src="opencv.js" type="text/javascript" defer></script>
        <script src="octree.js" type="text/javascript" defer></script>
        <script src="loadImg.js" type="text/javascript" defer></script>
        <title>Color Quantization</title>
    </head>
    <body>
        <span class="border -0">
            <h3 class="text-center">Color Quantization</h3>
            <br>
            <div class="container h-100">
                <div class="row justify-content-center ">
                    <div class="col-sm-8 align-self-center text-center">

                        <input type="file" id="file_input">
                        <br> <br>
                        <img id="input_image">
                        <br> <br>
                        <canvas id="output_image"></canvas>
                        <br> <br>
                        <canvas id="output_palette" width="4096" height="4096"></canvas>

                    </div>
                </div>
            </div>
        </span>
    </body>

<link rel="stylesheet" href="css/bootstrap.min.css">
<script src="js/bootstrap.min.js></script>
</html>
```

Código A.2: octree.js

```
function pad(n, width, z){
   z = z || '0';
   n = n + '';
   return n.length >= width ? n : new Array(width - n.length +1).join(z) + n;
}

function numberToBinaryString(n){
   let numberInString = Number((n >>> 0).toString(2));
   numberInString = pad(numberInString, 8);
   return numberInString;
}
```

```
12
13
14 class ArrayNodes{
15     constructor(){
16         this.array = [];
17     }
18     init(){
19         for(var i=0;i<8;i++){
20             let tempNode = new Node(true,null,null);
21             this.array.push(tempNode);
22         }
23     }
24 }
25
26 class Node{
27     constructor(leaf,color){
28         this.leaf=leaf;
29         this.color=color;
30         this.pixelCount= 0;
31         this.colorInRgb;
32         this.array= new ArrayNodes;
33         this.#isLeaf();
34     }
35
36     #isLeaf(){
37         if(!this.leaf){
38             this.array.init();
39         }
40     }
41
42     setLeaf(leaf){
43         this.leaf = leaf;
44         this.#isLeaf();
45     }
46
47     insertColor(colorRgb, colorInOctal, indice){
48         if(this.leaf){
49             this.pixelCount ++;
50             this.colorInRgb = colorRgb;
51         }
52         if(indice == 8)
53             return;
54         this.array.array[colorInOctal[indice]].color = colorInOctal[indice];
55         this.array.array[colorInOctal[indice]].insertColor(colorRgb, colorInOctal, indice + 1);
56
57     }
58
59     setColor(color, colorInOctal, indice){
60         if(this.leaf){
61             color.red = this.colorInRgb.red;
62             color.green = this.colorInRgb.green;
63             color.blue = this.colorInRgb.blue;
```

```
64              return;
65          }
66          this.array.array[colorInOctal[indice]].color = colorInOctal[indice];
67          this.array.array[colorInOctal[indice]].setColor(color, colorInOctal, indice + 1);
68      }
69
70      /*!*/
71      initHeight(height){
72          if(height==1){
73              for(let i = 0; i < 8; ++i){
74                  this.array.array[i].setLeaf(true);
75              }
76              return;
77          }
78          for(var i=0;i<8;i++){
79              this.array.array[i].setLeaf(false);
80              this.array.array[i].initHeight(height-1);
81          }
82      }
83  }
84
85  class Octree{
86      constructor(){
87          this.head = new Node(false,null,null);
88          this.height = 0;
89          this.head.initHeight(8);
90      }
91
92       fill (matriz){
93          //TODO change for size of matriz
94          for(let i = 0; i < matriz.length; ++i){
95              let red = numberToBinaryString(matriz[i].red);
96              let green = numberToBinaryString(matriz[i].green);
97              let blue = numberToBinaryString(matriz[i].blue);
98              let colorFinalTransformation = [];
99              for(let j = 0; j< 8; ++j){
100                 let octal = red[j] + green[j] + blue[j];
101                 let octalInDecimal = parseInt(octal, 2);
102                 colorFinalTransformation.push(octalInDecimal);
103             }
104             this.head.insertColor(matriz[i], colorFinalTransformation, 0);
105         }
106     }
107
108     reduction(node){
109         if(node.array.array[0].leaf == true ) {
110             var red = 0, green = 0, blue = 0, pixelCount = 0;
111
112             for(let i = 0; i < 8; ++i) {
113                 if(node.array.array[i].pixelCount === 0) {
114                     continue;
115                 }
```

```
116                 red += node.array.array[i].colorInRgb.red * node.array.array[i].pixelCount;
117                 green += node.array.array[i].colorInRgb.green * node.array.array[i].pixelCount;
118                 blue += node.array.array[i].colorInRgb.blue * node.array.array[i].pixelCount;
119                 pixelCount += node.array.array[i].pixelCount;
120             }
121 
122             node.leaf = true;
123             node.pixelCount = pixelCount;
124             if(pixelCount > 0 ) {
125                 var color = new Color( parseInt(red / pixelCount), parseInt(green / pixelCount),
    ↪ parseInt(blue / pixelCount ));
126                 node.colorInRgb = color;
127             }
128             node.array = [];
129 
130         } else {
131             for(let i = 0; i < 8; ++i) {
132                 this.reduction(node.array.array[i]);
133             }
134         }
135     }
136 
137     reconstruction(matriz){
138         for(let i = 0; i < matriz.length; ++i){
139             let red = numberToBinaryString(matriz[i].red);
140             let green = numberToBinaryString(matriz[i].green);
141             let blue = numberToBinaryString(matriz[i].blue);
142             let colorFinalTransformation = [];
143             for(let j = 0; j< 8; ++j){
144                 let octal = red[j] + green[j] + blue[j];
145                 let octalInDecimal = parseInt(octal, 2);
146                 colorFinalTransformation.push(octalInDecimal);
147             }
148             this.head.setColor(matriz[i], colorFinalTransformation, 0);
149 
150         }
151     }
152     pallete(nodo, mat){
153         if(nodo.leaf && nodo.pixelCount > 0){
154             mat.push(nodo.colorInRgb);
155             return;
156         }else if(nodo.leaf && nodo.color === null){
157             return;
158         }
159         for(let i =0; i < 8; ++i){
160             this.pallete(nodo.array.array[i], mat);
161         }
162     }
163 }
```

Código A.3: loadImg.js

```
let img_input = document.getElementById('input_image');
let file_input = document.getElementById('file_input');
let canvas = document.getElementById('output_image');
let canvasPalette = document.getElementById('output_palette');
let ctxCanvasPalette = canvasPalette.getContext('2d');
let ctx = canvas.getContext('2d');
let octree = new Octree();

file_input.addEventListener('change', (event) => {
    img_input.src = URL.createObjectURL(event.target.files[0]);
}, false);

class Color {
    constructor( red, green, blue ) {
        this.red = red;
        this.green = green;
        this.blue = blue;
    }
};

img_input.onload = function() {
    let mat = cv.imread(img_input);

    let img_width = img_input.naturalWidth;
    let img_height = img_input.naturalHeight;
    ctx.rect(0, 0, img_width, img_height);

    cv.imshow('output_image', mat);

    let data = ctx.getImageData(0, 0, img_width, img_height).data;
    let matrix_image = [];

    for(let i = 0; i < data.length; i = i + 4) {
        matrix_image.push(new Color(data[i], data[i+1], data[i+2]));
    }

    octree. fill (matrix_image);
    octree.reduction(octree.head);
    octree.reduction(octree.head);
    octree.reduction(octree.head);
    octree.reduction(octree.head);
    octree.reduction(octree.head);
    octree.reduction(octree.head);
    octree.reduction(octree.head);
    octree.reconstruction(matrix_image);

    console.log(octree);

    let count = 0;
    for(let i = 0; i < img_height; i++) {
        for(let j = 0; j < img_width; j++) {
            mat.ucharPtr(i, j)[0] = matrix_image[count].red;
```

```
53              mat.ucharPtr(i, j)[1] = matrix_image[count].green;
54              mat.ucharPtr(i, j)[2] = matrix_image[count].blue;
55              count++;
56          }
57      }
58      cv.imshow('output_image', mat);
59
60      var matRgbPalette = [];
61      octree.pallete(octree.head, matRgbPalette);
62
63      let sizeMatRgb = matRgbPalette.length;
64      console.log(sizeMatRgb);
65      let  sizePalette;
66      if(sizeMatRgb <= (1 << 18)){
67          sizePalette = 1 << 9;
68
69      }else{
70          sizePalette = 1 << 12;
71      }
72      console.log(sizePalette);
73      let matPalette = new cv.Mat(sizePalette, sizePalette, cv.CV_8UC3);
74      ctxCanvasPalette.rect(0, 0, sizePalette, sizePalette);
75
76      let count2 = 0, temp = 0;
77      let totalPixelsPerColor = Math.floor( sizePalette*sizePalette / sizeMatRgb);
78      console.log(totalPixelsPerColor);
79      for(let i = 0; i < sizePalette; ++i){
80          for(let j = 0; j < sizePalette; ++j){
81              if(count2 === sizeMatRgb){
82                  matPalette.ucharPtr(i, j)[0] = 255;
83                  matPalette.ucharPtr(i, j)[1] = 255;
84                  matPalette.ucharPtr(i, j)[2] = 255;
85                  continue;
86              }
87              matPalette.ucharPtr(i, j)[0] = matRgbPalette[count2].red;
88              matPalette.ucharPtr(i, j)[1] = matRgbPalette[count2].green;
89              matPalette.ucharPtr(i, j)[2] = matRgbPalette[count2].blue;
90
91              if(temp > totalPixelsPerColor - 1){
92                  count2++;
93                  temp = 0;
94              }
95              temp++;
96          }
97      }
98
99      cv.imshow('output_palette', matPalette);
100
101     mat.delete();
102 }
```

Anexo B. Capturas de pantalla

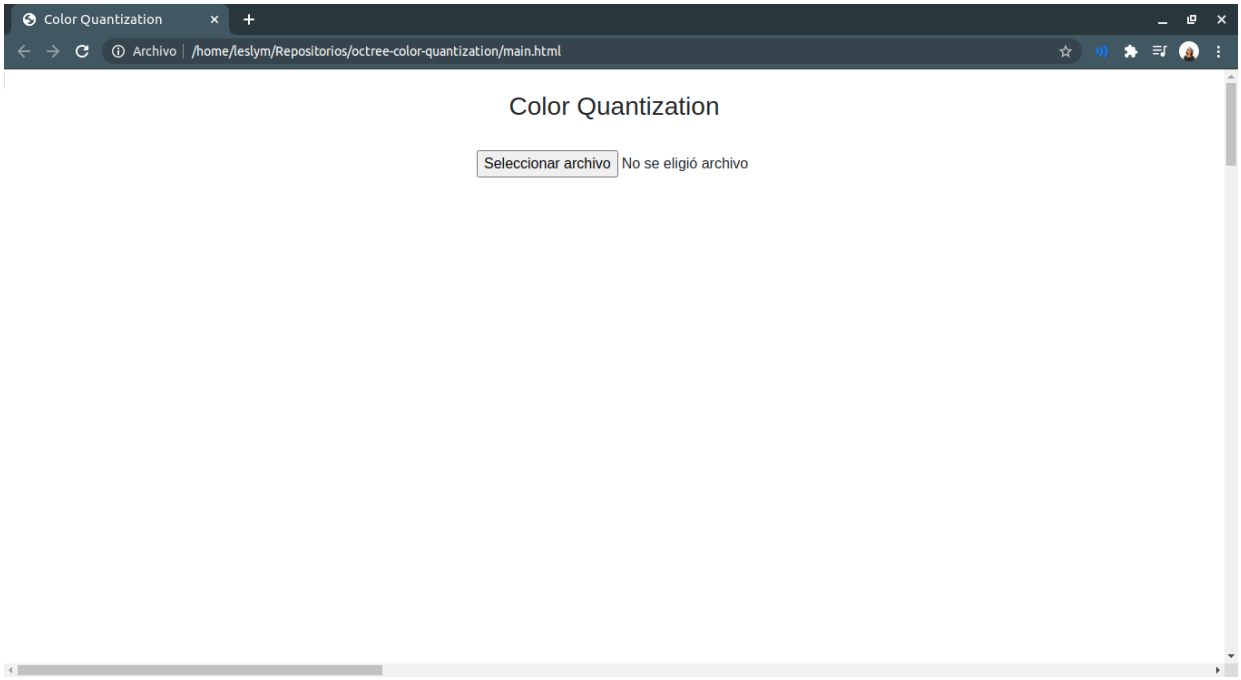


Figura B.1: Visualización 1

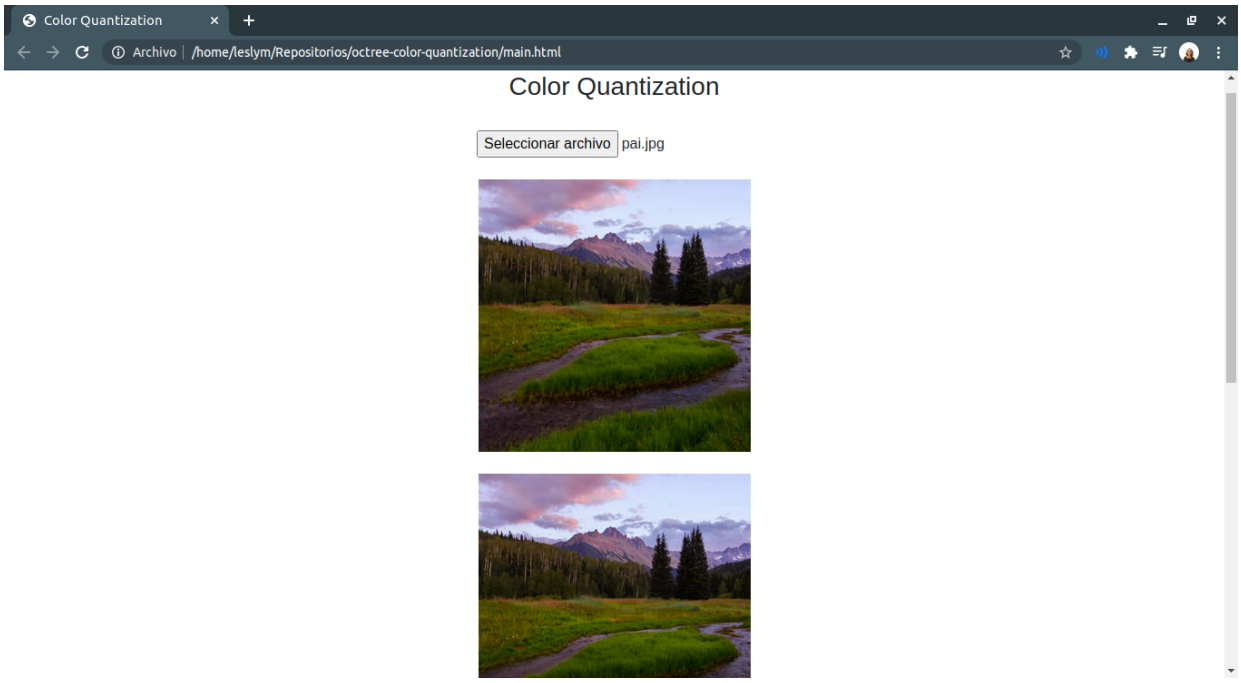


Figura B.2: Visualización 2

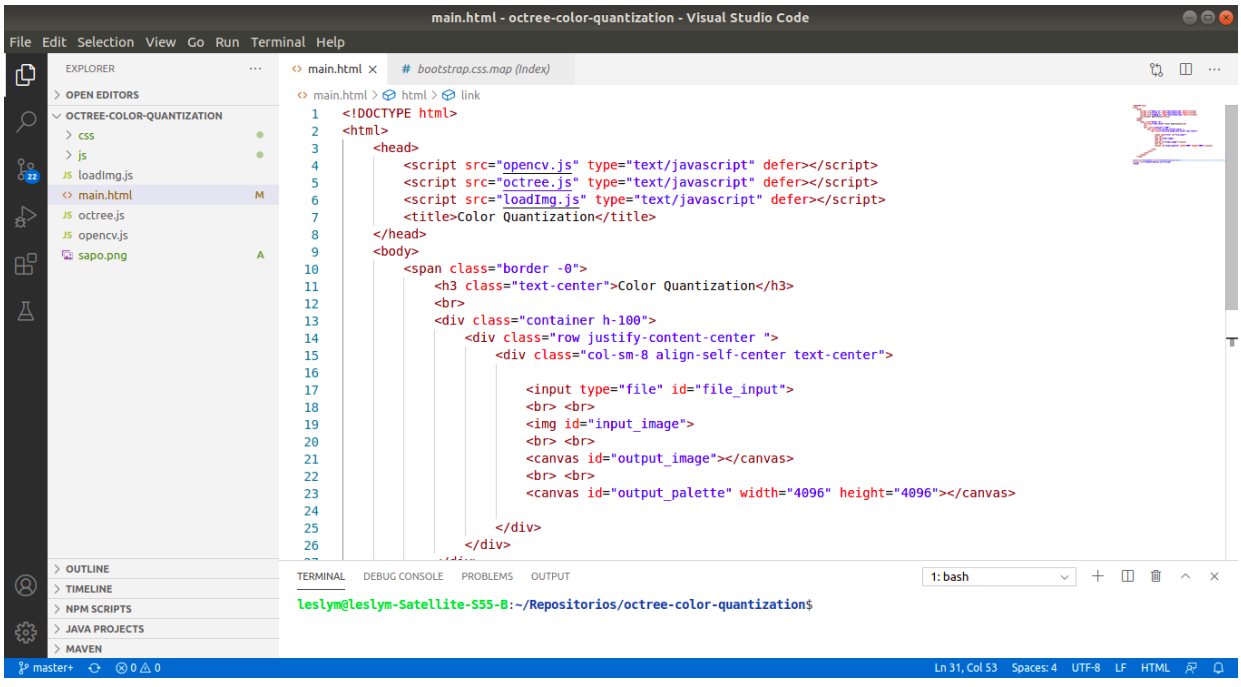


Figura B.3: Captura de Pantalla de main.html

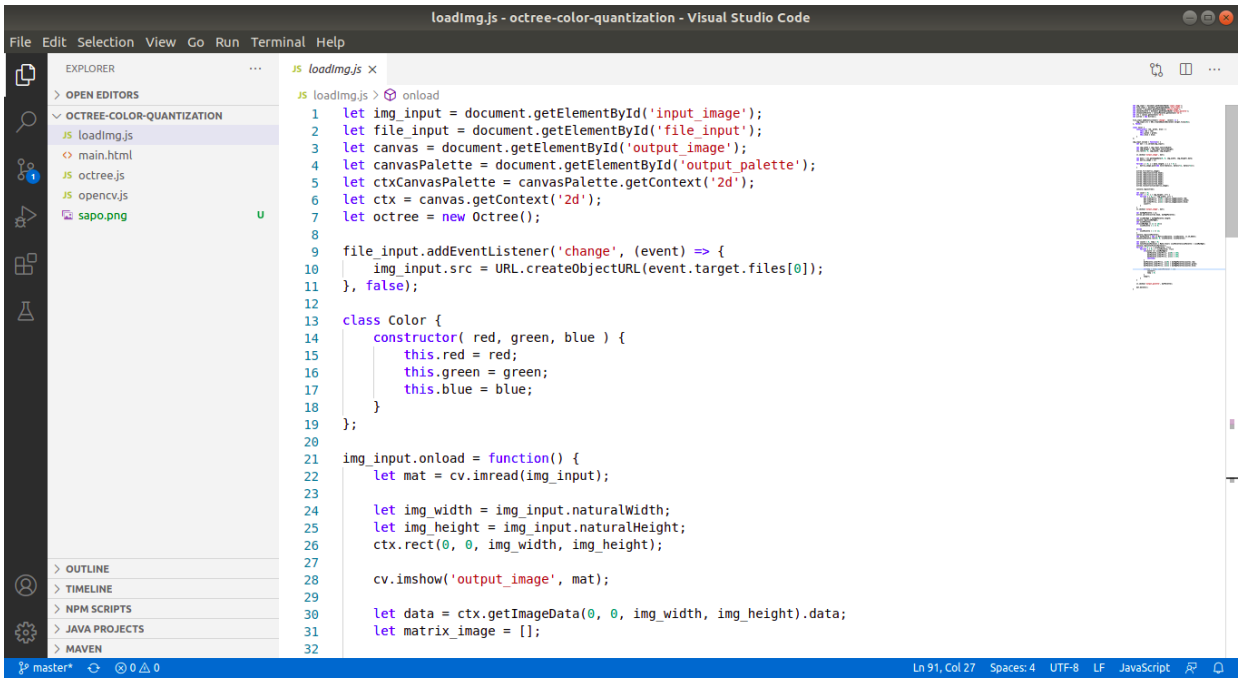


Figura B.4: Captura de Pantalla de loadImg.js

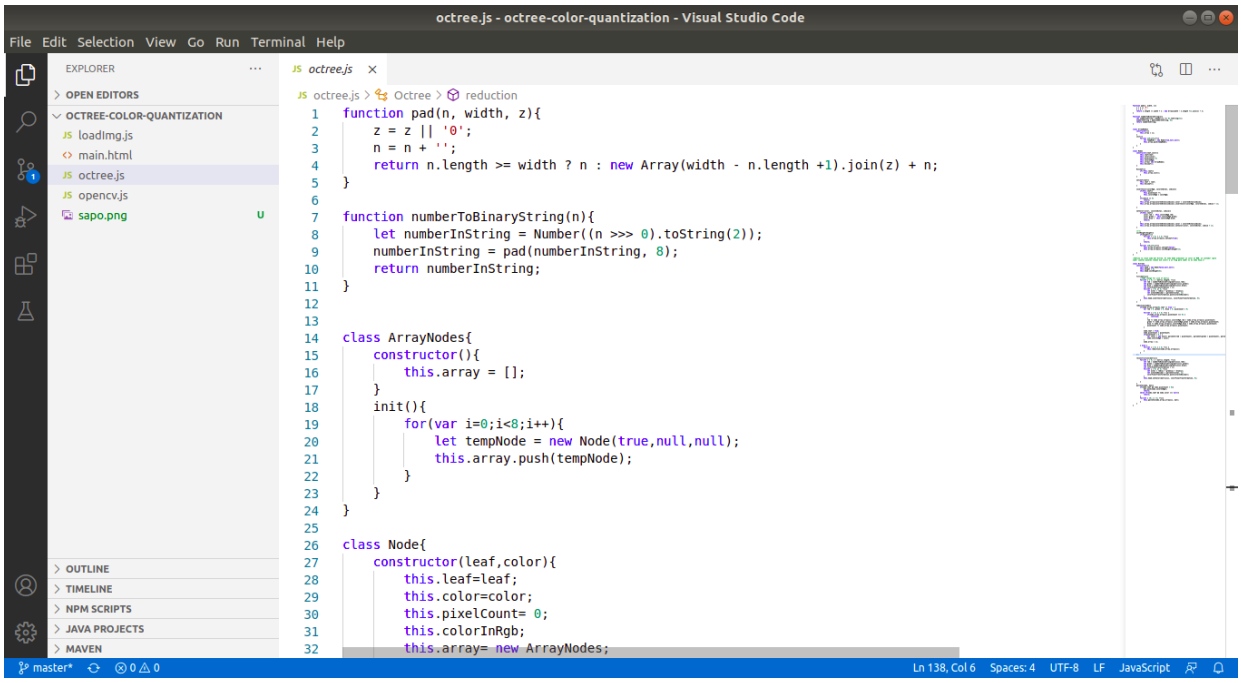


Figura B.5: Captura de Pantalla de octree.js

Anexo C. Repositorios

Toda la implementación de la estructura OcTree, el código, documento, imágenes y un README, se encuentran en los siguientes repositorios

- GitHub: https://github.com/Leslym03/octreeColorQuantization
- GitLab: https://gitlab.com/gabrieldgc1999/octree-color-quantization