UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

Facultad de Ingeniería de Producción y Servicios

Ciencia de la Computación

OcTree Color Quantization

ALUMNOS:
Barrionuevo Paredes, Fabricio José
Buendia Gutierrez, Ivan Rafael
Mamani Quispe, Alex
Panibra Mamani, Thales Gonzalo
Pfuturi Huisa, Oscar David
Tapara Quispe, Jhoel Salomon

DOCENTE:
Mg. Vicente Machaca Arceda

CURSO:
Estructuras de Datos Avanzadas

3 de noviembre de 2020

Índice

1. Introducción

El Octree es un árbol en el cual todo nodo padre tiene exactamente ocho nodos hijo. Estos nodos almacenarán la información de los píxeles de una imagen que están representadas en el modelo RGB. Para el Octree que se presentará en este documento, tendrá un máximo de ocho niveles, donde se distribuirán los píxeles, además de una paleta que contenga los colores disponibles dentro del árbol.

2. QNode

Utilizamos nodos para almacenar los colores de cada píxel de la imagen.

2.1. Constructor:

Al instanciar un nodo, este recibe como parámetro un número que representa el nivel donde será creado dicho nodo, asi mismo se crea un arreglo que almacenará ocho nodos (hijos), un booleano que representa si un nodo es hoja o no, un objeto del tipo acumulador de píxeles y un contador.

```
class QNode {
  pixelAccumulator: RGBPixelAccumulator;
  pixelCount: number;
  children: QNode[];
  leaf: boolean;
  level: number;
  constructor(level: number) {
    this.level = level;
    this.leaf = true;
    this.pixelAccumulator = new RGBPixelAccumulator({ r: 0, g: 0, b: 0 });
    this.children = Array<QNode>(8);
    this.pixelCount = 0;
  }
  ...
}
```

2.2. Insertar píxel:

Para insertar un píxel dentro de los nodos del Octree, se requiere de los valores del píxel y el nivel al cual pertenece. La función trabaja usando la recursividad, una vez que se inserta un píxel, se procede a insertar el siguiente valor al próximo nivel.

```
  insertRGBPixel(rgbPixel: RGBPixel, level: number): void {
    if (level >= LEVELS) {
      this.pixelAccumulator.sumRGBValues(rgbPixel);
      this.pixelCount++;
    } else {
      this.leaf = false;
      let index = getPixelIndex(rgbPixel, level);
      // not null
      if (!this.children[index]) {
        this.children[index] = new QNode(level);
      }
      this.children[index].insertRGBPixel(rgbPixel, level + 1);
    }
  }
```

3. Acumulador RGB

Clase que permite guardar todos los colores existentes dentro de una imagen.

3.1. Métodos

El constructor asigna un color a su único atributo.
Contiene un método el cual sirve para acumular los valores de los colores del último nivel al momento en que se reduzcan los niveles del árbol.

```
class RGBPixelAccumulator {
  baseRGB: RGBPixel;
  constructor(rgbPixel: RGBPixel) {
    this.baseRGB = rgbPixel;
  }
  sumRGBValues(rgbPixel: RGBPixel): void {
    this.baseRGB.r += rgbPixel.r;
    this.baseRGB.g += rgbPixel.g;
    this.baseRGB.b += rgbPixel.b;
  }
}
```

4. Funcionalidades:

4.1. RBGPixel:

Almacena la composición de un color en términos del modelo RGB.

```
interface RGBPixel {
  r: number;
  g: number;
  b: number;
}
```

4.2. getPixelIndex:

Esta función devuelve el índice del nodo hijo en el que se debe insertar el valor del modelo RGB por cada nivel, completando de esa forma los ocho niveles del Octree. La operación para obtener el índice es a nivel de bits, se calcula los valores en base decimal de los tres componentes RGB bit a bit, de modo que obtenga los índices de los nodos, como los colores solo pueden tener hasta ocho bits cada uno, utilizamos una máscara de bits y las desplazamos n ceros a la derecha donde n es el nivel al que pertenece dicho bit y se procede a utilizar el operador OR con los números 4, 2 y 1 para calcular dicho índice.

```
const getPixelIndex = (rgbPixel: RGBPixel, level: number): number => {
  let index = 0;
  let mask = 128 >> level;
  if (rgbPixel.r & mask) index |= 4;
  if (rgbPixel.g & mask) index |= 2;
  if (rgbPixel.b & mask) index |= 1;
  return index;
};
```

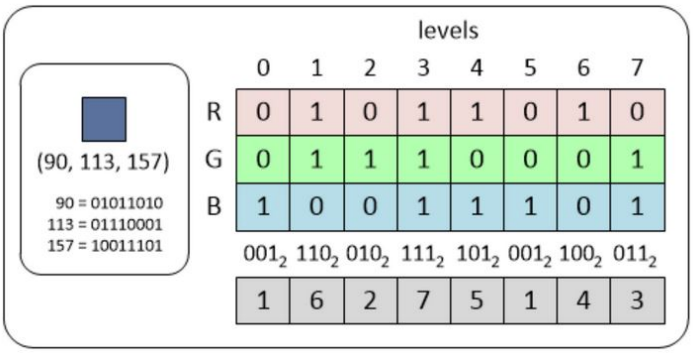

Figura 1

5. QuantizerOctree

5.1. Constructor:

Para la construcción del Octree inicializamos un nodo raíz, dos arreglos que almacenarán la imagen de entrada y salida, sus dimensiones y el número máximo de niveles que tiene el árbol. La raíz se considera de nivel -1 para evitar errores al momento de insertar elementos dentro del árbol.

```
export class QuantizerOctree {
  //   levels: number[];
  private root: QNode;
  private outputImg: Uint8ClampedArray; // 0 - 255
  private inputImg: Uint8ClampedArray;
  private imgHeight = -1;
  private imgWidth = -1;
  private max_level: number;
  constructor() {
    // this.levels = new Array<number>(LEVELS);
    this.root = new QNode(-1);
    this.outputImg = new Uint8ClampedArray();
    this.inputImg = new Uint8ClampedArray();
    this.max_level = 7;
  }
  ...
  ...
  ...
}
```

5.2. Insertar un Píxel:

Consiste en recibir como parámetro la información de un píxel e invocar a la propia función del nodo para realizar la inserción.

```
private insertPixel(color: RGBPixel): void {
    this.root.insertRGBPixel(color, 0);
}
```

5.3. Imagen de salida:

Para crear la imagen modificada primero se debe crear un ciclo en el cual buscaremos los nodos hoja de cada píxel de la imagen dentro del árbol. Una vez se halle el nodo hoja, modificaremos los valores

del acumulador de píxeles dividiéndolos entre el contador del nodo, y estos valores serán asignados al arreglo de la imagen de salida.

```
1   private buildOutputImg(): void {
2     this.outputImg = new Uint8ClampedArray(4 * this.imgWidth * this.
    imgHeight);
3     for (let i = 0; i < this.inputImg.length; i += 4) {
4       let rgbaPixel: RGBAPixel = {
5         r: this.inputImg[i + 0],
6         g: this.inputImg[i + 1],
7         b: this.inputImg[i + 2],
8         a: this.inputImg[i + 3]
9       };
10      let rgbPixel: RGBPixel = {
11        r: this.inputImg[i + 0],
12        g: this.inputImg[i + 1],
13        b: this.inputImg[i + 2]
14      };
15      let qnode = this.getQNode(rgbPixel, this.root, 0);
16      let outputPixel: RGBPixel = {
17        r: qnode.pixelAccumulator.baseRGB.r / qnode.pixelCount,
18        g: qnode.pixelAccumulator.baseRGB.g / qnode.pixelCount,
19        b: qnode.pixelAccumulator.baseRGB.b / qnode.pixelCount
20      };
21      this.outputImg[i + 0] = outputPixel.r;
22      this.outputImg[i + 1] = outputPixel.g;
23      this.outputImg[i + 2] = outputPixel.b;
24      this.outputImg[i + 3] = rgbaPixel.a;
25    }
26  }
```

5.4. Reducción de niveles:

La función *reduceLevel* recibe como parámetro la cantidad de niveles a reducir y utilizará un ciclo para realizar la reducción. Se creará un array temporal "qchild"que almacenará el mapeo de las hojas del octree, y se hace la reducción a través del llamado a la función "sumRGBValues"del objeto QNode donde acumulará los valores del nodo hijo al nodo padre. Al mismo tiempo el booleano "leaf"del padre pasará a ser verdadera.

```
1   private BFS_reduce(root: QNode): void {
2     this.max_level--;
3     let queue = [root];
4     let visited = new Set<QNode>();
5     while (queue.length > 0) {
6       let qnode = queue.shift()!;
7       for (let qchild of qnode.children) {
8         if (qchild) {
9           if (qnode.level === this.max_level) {
10            qnode.leaf = true;
11            qnode.pixelCount += qchild.pixelCount;
12            qnode.pixelAccumulator.sumRGBValues(
13              qchild.pixelAccumulator.baseRGB
```

```
            );
          }
          if (!visited.has(qchild) && qchild.level <= this.max_level) {
            visited.add(qchild);
            queue.push(qchild);
          }
        }
      }
    }
  }

  private reduceLevel(levelsToReduce: number): void {
    if (levelsToReduce > LEVELS) levelsToReduce = LEVELS;
    for (let i = 0; i < levelsToReduce; i++) {
      this.BFS_reduce(this.root);
    }
  }
```

5.5. getQnode:

La siguiente función recorrerá los nodos de forma recursiva. Utilizando un ciclo, buscará nivel por nivel hasta encontrar el camino de nodos hasta el nodo hoja que le corresponde al píxel de entrada. Esta función se llama en la función "buildOutputImg".

```
private getQNode(rgbPixel: RGBPixel, root: QNode, level: number):
 QNode {
  while (!root.leaf) {
    const index = getPixelIndex(rgbPixel, level);
    root = root.children[index];
    level++;
  }
  return root;
}
```

5.6. fill:

El método fill nos ayuda a leer los píxeles de la imagen y guardarlos en la interfaz "RGBpixel".

```
fill(data: Uint8ClampedArray, width: number, height: number): void {
  console.log({
    newWidth: width,
    newHeight: height,
  });
  console.log({ length: data.length });
  this.imgWidth = width;
  this.imgHeight = height;
  this.inputImg = data;
  for (let i = 0; i < data.length; i += 4) {
    let rgbPixel: RGBPixel = {
      r: data[i + 0],
      g: data[i + 1],
      b: data[i + 2],
    };
    this.insertPixel(rgbPixel);
  }
}
```

5.7. reduceAndShow:

La siguiente función carga una imagen del HTML, y se hace un llamado a la función "reduceLeve"donde usaremos "BFS_reduce"mencionado anteriormente.

```
reduceAndShow(canvasId: string, levelsToReduce: number): void {
  const canvas = document.getElementById(canvasId) as HTMLCanvasElement;
  canvas.width = this.imgWidth;
  canvas.height = this.imgHeight;
  const ctx = canvas.getContext("2d")!;
  this.reduceLevel(levelsToReduce);
  this.buildOutputImg();
  let imageData = new ImageData(
    this.outputImg,
    this.imgWidth,
    this.imgHeight
  );
  // Draw image data to the canvas
  ctx.putImageData(imageData, 0, 0);
}
```

6. Cargar imagen

Importamos primero el objeto "QuantizerOctree"del archivo "colored-octree.ts". Luego a través de la variable "target"procedemos a leer la imagen del cargada en el html tomada como una secuencia en FileList. Cada que exista un archivo lo tomará como una imagen en la variable "img".
La función "resizereduce el tamaño de la imagen en caso sea muy grande.
La función "fillWithImgrecibirá los parámetros necesarios para llamar a la función fill del Octree.
Entonces, en general, este archivo llega a capturar el evento para luego leer la imagen. Y este se sobrecarga.

```
import { QuantizerOctree } from "./colored-octree";

export const loadImg = (e: Event, octree: QuantizerOctree) => {
  let target = e.target as HTMLInputElement;
  let files = target.files as FileList;
  if (files) {
    let imagefile = files.item(0)!;
    let reader = new FileReader();
    reader.readAsDataURL(imagefile);

    reader.onloadend = (ev: Event) => {
      let img = new Image();
      img.src = reader.result as string;
      img.onload = (ev: Event) => {
        let canvas = document.getElementById("input-img") as HTMLCanvasElement;
        let context = canvas.getContext("2d") as CanvasRenderingContext2D;

        // original image size
        // canvas.width = img.width;
        // canvas.height = img.height;
        // context.drawImage(img, 0, 0);

        // resize
        resize(canvas, context, img);

        console.log({
```

```
          rawWidth: img.width,
          rawHeight: img.height,
        });

        // QOctree
        fillWithImg(canvas, octree);
        showResult(octree);
      };
    };
  }
};

const resize = (
  canvas: HTMLCanvasElement,
  context: CanvasRenderingContext2D,
  img: HTMLImageElement
) => {
  canvas.height = canvas.width * (img.height / img.width);
  let oc = document.createElement("canvas");
  let ocontext = oc.getContext("2d") as CanvasRenderingContext2D;
  oc.width = img.width * 0.5;
  oc.height = img.height * 0.5;
  ocontext.drawImage(img, 0, 0, oc.width, oc.height);
  ocontext.drawImage(oc, 0, 0, oc.width * 0.5, oc.height * 0.5);
  context.drawImage(
    oc,
    0,
    0,
    oc.width * 0.5,
    oc.height * 0.5,
    0,
    0,
    canvas.width,
    canvas.height
  );
};

const fillWithImg = (canvas: HTMLCanvasElement, octree: QuantizerOctree) => {
  const context = canvas.getContext("2d") as CanvasRenderingContext2D;
  const imgData = context.getImageData(0, 0, canvas.width, canvas.height);
  const data = imgData.data;
  octree.fill(data, canvas.width, canvas.height);
};

const showResult = (octree: QuantizerOctree) => {
  const element = document.getElementById("levels-to-reduce") as
    HTMLSelectElement;
  const options = element.options[element.selectedIndex];
  const value = parseInt(options.value);
  octree.reduceAndShow("output-img", value);
};
```

7. index.ts

El archivo “index.ts”nos ayuda a importar la imagen del html, y a su vez llama al archivo “colored-octree.ts”quien contiene el objeto “QuantizerOctree”para hacer la divisió de nodos; y llama al archivo “imgloader”quien tiene al objeto “loadImg”quien también trabaja con el “QuantizerOctree”. Luego

se asignará memoria para este nuevo Octree y se procederá a ejecutar los mapeos.

```
import { QuantizerOctree } from "./colored-octree";
import { loadImg } from "./imgloader";

let imgInput = document.getElementById("imageInput") as HTMLInputElement;
let octree: QuantizerOctree;
imgInput.addEventListener("change", (e: Event) => {
  octree = new QuantizerOctree();
  console.log(octree);
  loadImg(e, octree);
});
```

8. Front

Se tendrá además un archivo html que permitirá cargar una imagen, y la reducción será a elección del usuario. Tendrá un desplegable de 9 opciones.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <div>
      <input type="file" id="imageInput" accept="image/*" />
      <br />
      <h>Levels to reduce</h>
      <select id="levels-to-reduce">
        <option value="0">0</option>
        <option value="1">1</option>
        <option value="2">2</option>
        <option value="3">3</option>
        <option value="4">4</option>
        <option value="5">5</option>
        <option value="6">6</option>
        <option value="7">7</option>
        <option value="8">8</option>
      </select>
      <br />
      <canvas id="input-img" width="500"></canvas>
      <canvas id="output-img"></canvas>
    </div>
    <div id="app"></div>
  </body>
</html>
```

9. Pruebas

9.1. Elección de imagen y nivel

Figura 2

9.2. Imagen cargada

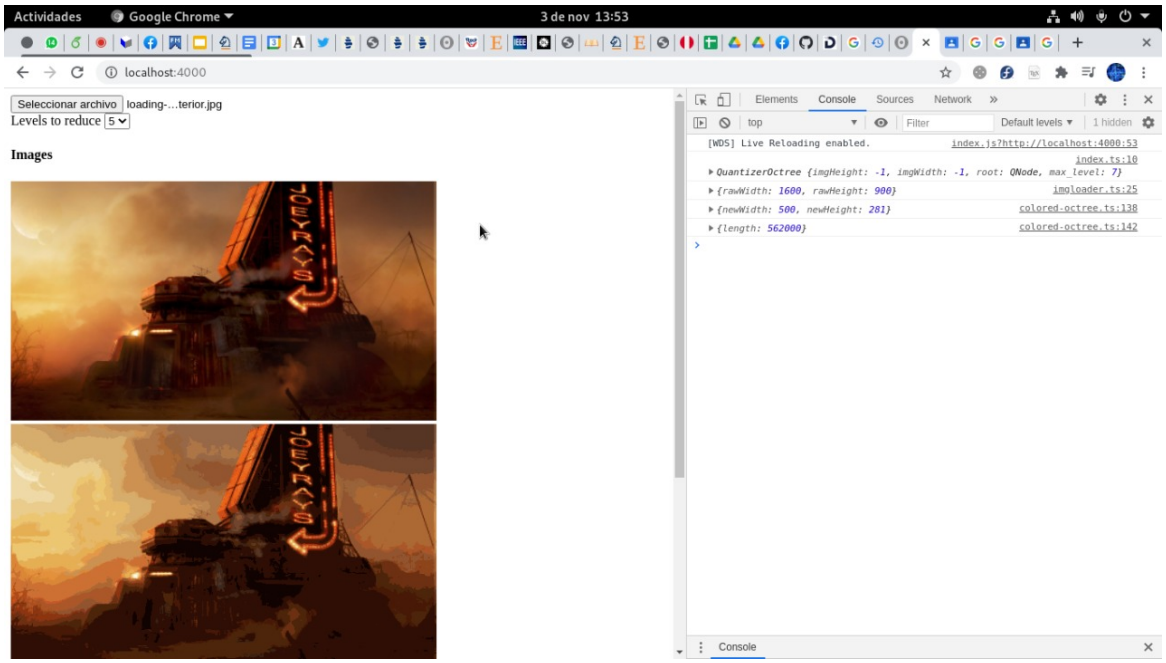

Figura 3

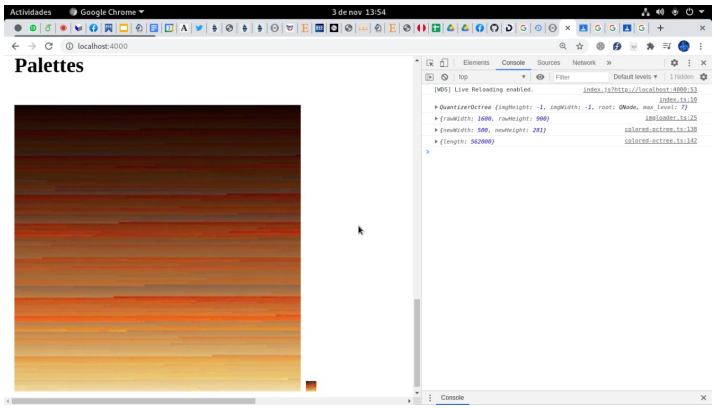

Figura 4

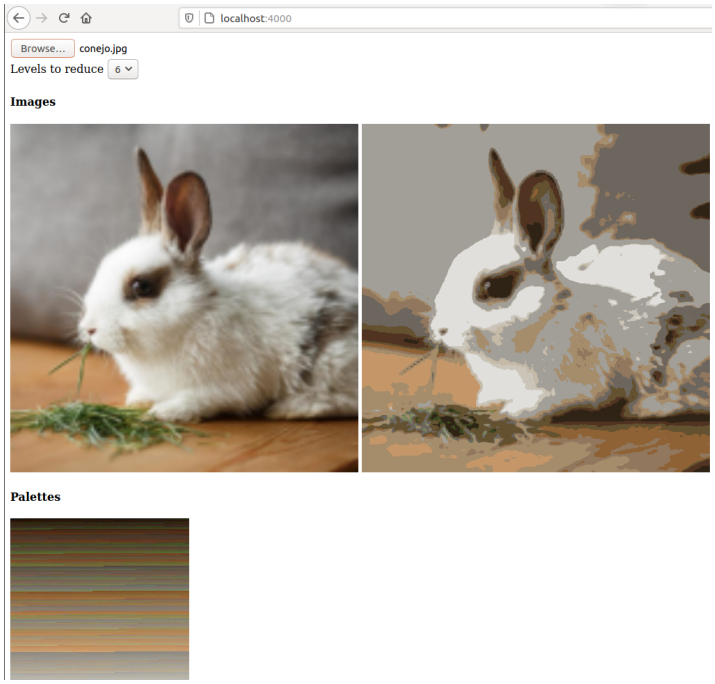

Figura 5

10. Enlaces

El código fuente puede encontrarse en el siguiente enlace:
https://github.com/Fabricio-Jose/Grupo-4-5-EDA/tree/typescript