**EJERCICIO QUAD TREES**

**Ejercicio 1: Descomposición de Imágenes con Quad Trees**

Definición de algoritmo “Quad Tree” que maneja la descomposición de la imagen y la reconstrucción con el lenguaje de programación JavaScript.

1. **Se definen variables y se asignan eventos para cargar la imagen y ajustar el umbral.**

<script>

*const* canvasOriginal = document.getElementById('canvas-original');

*const* ctxOriginal = canvasOriginal.getContext('2d');

*const* canvasReconstruido = document.getElementById('canvas-reconstruido');

*const* ctxReconstruido = canvasReconstruido.getContext('2d');

*let* umbralVarianza = 500; // Valor inicial del umbral

*let* archivo = null; // Almacena el archivo de imagen cargado

*const* cargarImagen = (*evento*) *=>* {

            archivo = evento.target.files[0];

*const* lector = new FileReader();

            lector.onload = (*e*) *=>* {

*const* imagen = new Image();

                imagen.onload = () *=>* {

                    canvasOriginal.width = imagen.width;

                    canvasOriginal.height = imagen.height;

                    canvasReconstruido.width = imagen.width;

                    canvasReconstruido.height = imagen.height;

                    ctxOriginal.drawImage(imagen, 0, 0);

*const* datosImagen = ctxOriginal.getImageData(0, 0, canvasOriginal.width, canvasOriginal.height);

*const* quadTree = new QuadTree(datosImagen, 0, 0, canvasOriginal.width, canvasOriginal.height);

                    quadTree.descomponer(umbralVarianza);

*const* datosReconstruidos = ctxReconstruido.createImageData(datosImagen.width, datosImagen.height);

                    quadTree.reconstruir(datosReconstruidos, umbralVarianza); // Pasamos el umbral para reconstruir

                    ctxReconstruido.putImageData(datosReconstruidos, 0, 0);

                    quadTree.dibujar(ctxReconstruido);

                };

                imagen.src = e.target.result;

            };

            lector.readAsDataURL(archivo);

        };

        document.getElementById('cargar').addEventListener('change', cargarImagen);

*const* umbralSlider = document.getElementById('umbral-slider');

        umbralSlider.addEventListener('input', () *=>* {

            umbralVarianza = parseInt(umbralSlider.value);

            if (archivo) {

                cargarImagen({ target: { files: [archivo] } });

            }

        });

1. **Inicializa el QuadTree con datos de la imagen y coordenadas.**

*class* QuadTree {

*constructor*(*datosImagen*, *x*, *y*, *ancho*, *alto*) {

                this.datosImagen = datosImagen;

                this.x = x;

                this.y = y;

                this.ancho = ancho;

                this.alto = alto;

                this.hijos = [];

                this.valorMedio = null;

            }

1. **Calcula el valor medio de los colores en el área actual.**

obtenerValorMedio() {

                if (this.valorMedio !== null) return this.valorMedio;

*const* datos = this.datosImagen.data;

*let* sumaR = 0, sumaG = 0, sumaB = 0, contador = 0;

                for (*let* j = this.y; j < this.y + this.alto; j++) {

                    for (*let* i = this.x; i < this.x + this.ancho; i++) {

*const* indice = (j \* this.datosImagen.width + i) \* 4;

                        sumaR += datos[indice];

                        sumaG += datos[indice + 1];

                        sumaB += datos[indice + 2];

                        contador++;

                    }

                }

                this.valorMedio = [

                    sumaR / contador,

                    sumaG / contador,

                    sumaB / contador

                ];

                return this.valorMedio;

            }

1. **Calcula la varianza de los colores en el área actual.**

obtenerVarianza() {

*const* datos = this.datosImagen.data;

*const* media = this.obtenerValorMedio();

*let* sumaCuadradosR = 0, sumaCuadradosG = 0, sumaCuadradosB = 0, contador = 0;

                for (*let* j = this.y; j < this.y + this.alto; j++) {

                    for (*let* i = this.x; i < this.x + this.ancho; i++) {

*const* indice = (j \* this.datosImagen.width + i) \* 4;

                        sumaCuadradosR += (datos[indice] - media[0]) \*\* 2;

                        sumaCuadradosG += (datos[indice + 1] - media[1]) \*\* 2;

                        sumaCuadradosB += (datos[indice + 2] - media[2]) \*\* 2;

                        contador++;

                    }

                }

                return (sumaCuadradosR + sumaCuadradosG + sumaCuadradosB) / contador;

            }

1. **Descompone el área en sub-áreas si la varianza es mayor que el umbral.**

descomponer(*umbral*) {

                if (this.obtenerVarianza() > umbral && this.ancho > 1 && this.alto > 1) {

*const* mitadAncho = Math.floor(this.ancho / 2);

*const* mitadAlto = Math.floor(this.alto / 2);

                    this.hijos.push(

                        new QuadTree(this.datosImagen, this.x, this.y, mitadAncho, mitadAlto),

                        new QuadTree(this.datosImagen, this.x + mitadAncho, this.y, mitadAncho, mitadAlto),

                        new QuadTree(this.datosImagen, this.x, this.y + mitadAlto, mitadAncho, mitadAlto),

                        new QuadTree(this.datosImagen, this.x + mitadAncho, this.y + mitadAlto, mitadAncho, mitadAlto)

                    );

                    this.hijos.forEach(*hijo* *=>* hijo.descomponer(umbral));

                }

            }

1. **Dibuja los límites del Quad Tree en el lienzo.**

dibujar(*ctx*) {

                if (this.hijos.length === 0) {

                    ctx.strokeStyle = 'black';

                    ctx.strokeRect(this.x, this.y, this.ancho, this.alto);

                } else {

                    this.hijos.forEach(*hijo* *=>* hijo.dibujar(ctx));

                }

            }

1. **Reconstruye la imagen procesada basándose en el umbral.**

reconstruir(*datosImagen*, *umbral*) {

                if (this.hijos.length === 0 || this.obtenerVarianza() <= umbral) {

*const* media = this.obtenerValorMedio();

*const* alpha = 255; //

                    // Opacidad máxima

*const* datos = datosImagen.data;

                    for (*let* j = this.y; j < this.y + this.alto; j++) {

                        for (*let* i = this.x; i < this.x + this.ancho; i++) {

*const* indice = (j \* datosImagen.width + i) \* 4;

                            datos[indice] = media[0];

                            datos[indice + 1] = media[1];

                            datos[indice + 2] = media[2];

                            datos[indice + 3] = alpha;

                        }

                    }

                } else {

                    this.hijos.forEach(*hijo* *=>* hijo.reconstruir(datosImagen, umbral));

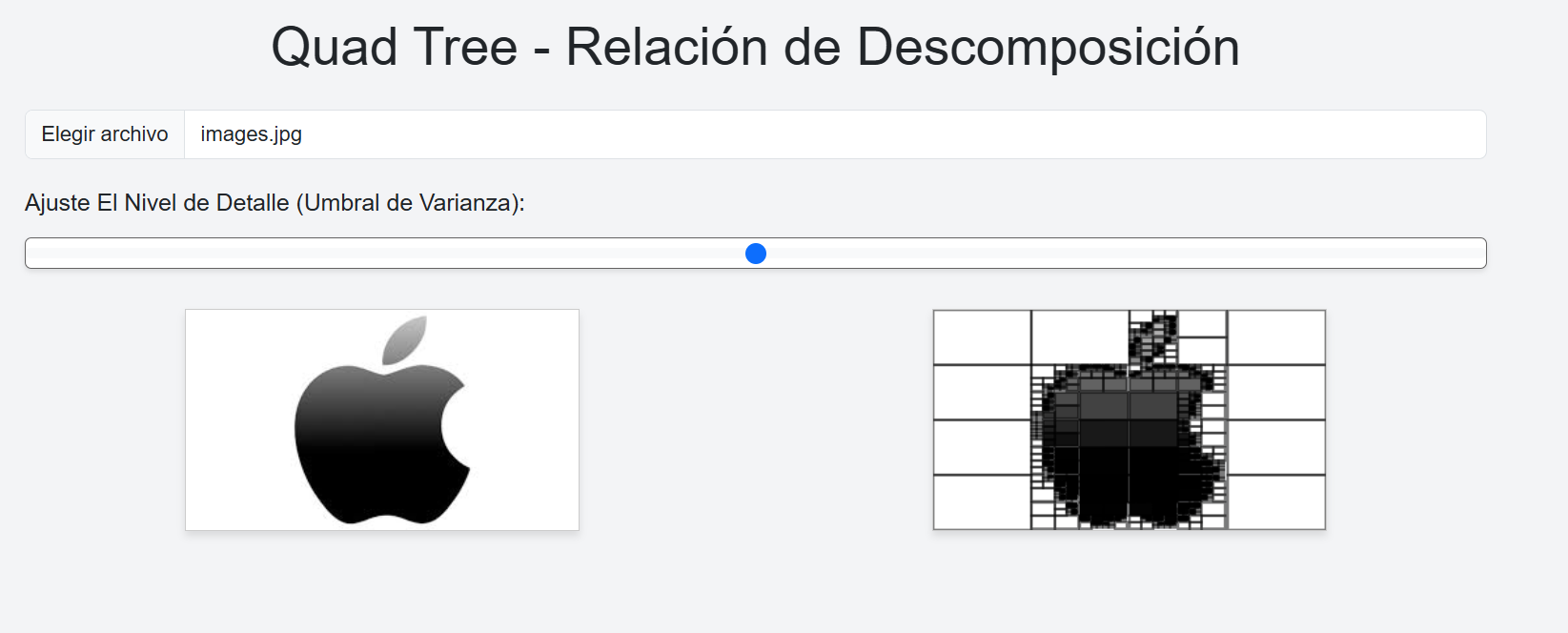
                }

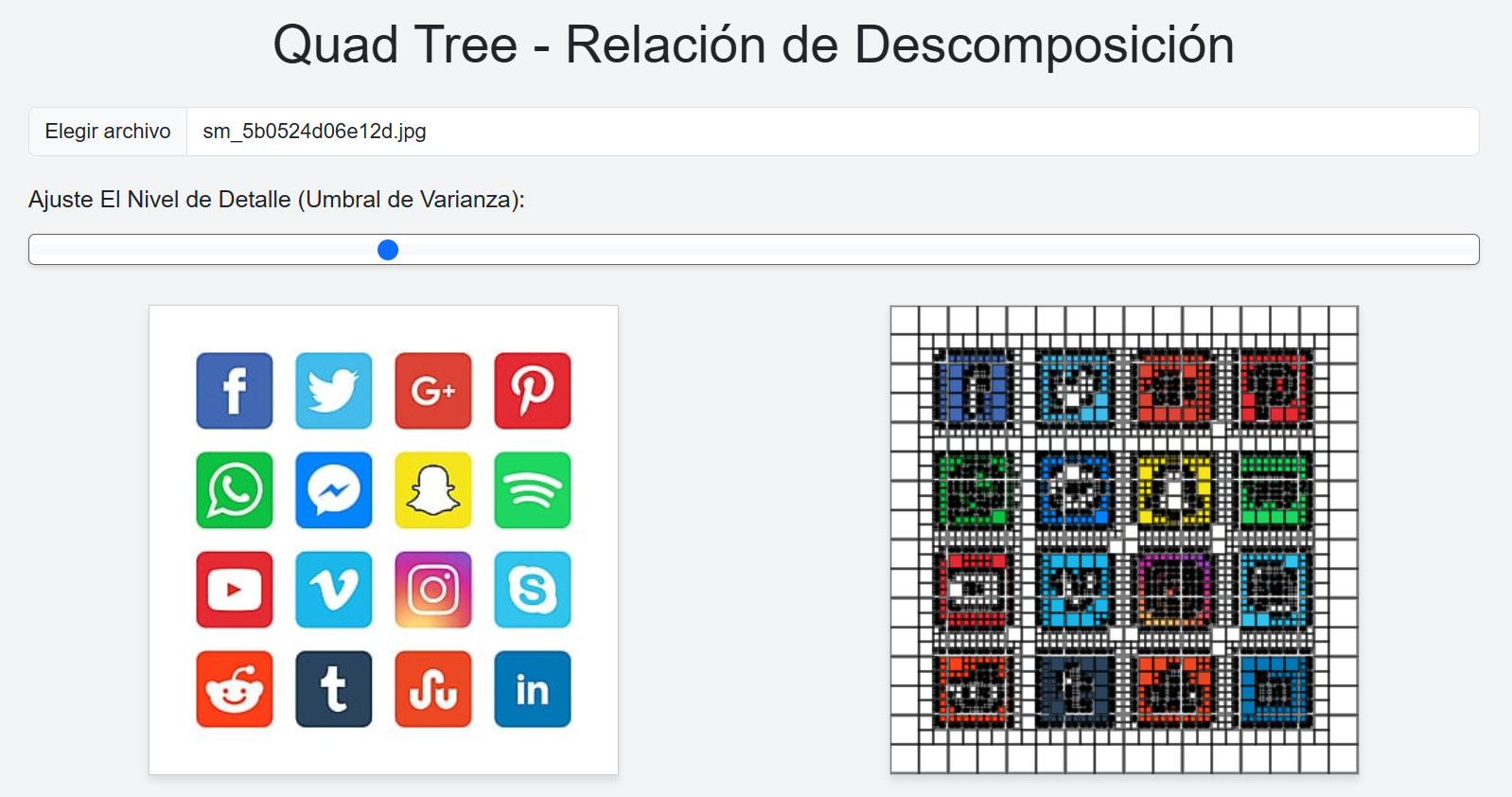
            }

        }

</script>

**Resultados:**

**Imagen N° 01:** Logotipo de Apple

** Imagen N° 02:** Logotipos deRedes Sociales

**Imagen N° 03:** Puerta Principal de la UNA-PUNO

**Ejercicio 2: Compresión de Imágenes con Quad Trees**

* **Añadimos esta función de comprimir**

comprimir() {

                if (this.obtenerVarianza() <= 100 || this.ancho <= 1 || this.alto <= 1 || this.hijos.length >= this.maxDescomposicion) {

                    // Umbral de varianza para la compresión o límite de descomposición alcanzado

*const* media = this.obtenerValorMedio();

                    for (*let* j = this.y; j < this.y + this.alto; j++) {

                        for (*let* i = this.x; i < this.x + this.ancho; i++) {

*const* indice = (j \* this.datosImagen.width + i) \* 4;

                            this.datosImagen.data[indice] = media[0];

                            this.datosImagen.data[indice + 1] = media[1];

                            this.datosImagen.data[indice + 2] = media[2];

                        }

                    }

                    this.hijos = []; // Limpiar hijos ya que esta región está comprimida

                } else {

*const* mitadAncho = Math.floor(this.ancho / 2);

*const* mitadAlto = Math.floor(this.alto / 2);

                    this.hijos.push(

                        new QuadTree(this.datosImagen, this.x, this.y, mitadAncho, mitadAlto),

                        new QuadTree(this.datosImagen, this.x + mitadAncho, this.y, mitadAncho, mitadAlto),

                        new QuadTree(this.datosImagen, this.x, this.y + mitadAlto, mitadAncho, mitadAlto),

                        new QuadTree(this.datosImagen, this.x + mitadAncho, this.y + mitadAlto, mitadAncho, mitadAlto)

                    );

                    this.hijos.forEach(*hijo* *=>* hijo.comprimir());

                }

            }

Ejemplo N° 01