بسم الله الرحمن الرحيم

|  |  |
| --- | --- |
| Name | Mahmoud Darwish |
| NIM | 09021081924166 |
| Class | 5 Bil A |

Contents

[Screenshots 1](#_Toc82465838)

[Start 1](#_Toc82465839)

[Vertical Edge Detection 2](#_Toc82465840)

[Horizontal Edge Detection 2](#_Toc82465841)

[Sharpen 3](#_Toc82465842)

[Blur 3](#_Toc82465843)

[Emboss 4](#_Toc82465844)

[The Code 4](#_Toc82465845)

# Screenshots

## Start

A screenshot of a computer

Description automatically generated with low confidence

## Vertical Edge Detection

A picture containing graphical user interface

Description automatically generated

## Horizontal Edge Detection

A picture containing graphical user interface

Description automatically generated

## Sharpen

A group of white flowers

Description automatically generated with low confidence

## Blur

A screenshot of a computer

Description automatically generated with low confidence

## Emboss

A screenshot of a computer

Description automatically generated with low confidence

# The Code

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <!-- Bootstrap CSS -->

    <link

      href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/css/bootstrap.min.css"

      rel="stylesheet"

      integrity="sha384-F3w7mX95PdgyTmZZMECAngseQB83DfGTowi0iMjiWaeVhAn4FJkqJByhZMI3AhiU"

      crossorigin="anonymous"

    />

    <title>Convolution by Mahmoud Darwish</title>

    <style></style>

  </head>

  <body

    class="

      d-flex

      flex-column

      align-items-center

      gap-1

      p-2

      bg-success bg-opacity-50

    "

  >

    <!-- Title -->

    <h1 class="display-1 p-3 m-0">بسم الله الرحمن الرحيم</h1>

    <!-- The URL input box -->

    <div class="d-flex align-self-stretch gap-1">

      <label class="display-6" for="imageUrl">Image URL</label>

      <input

        class="align-self-stretch flex-fill"

        type="url"

        name="imageUrl"

        id="imageUrl"

        placeholder="https://example.org/pic.png"

        value="https://images.unsplash.com/photo-1627878918787-b2cc3094659c?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1050&q=80"

      />

    </div>

    <!-- The canvases -->

    <div class="h-50 d-flex p-1 gap-1">

      <canvas id="cnvx0" class="img-fluid w-50"></canvas>

      <canvas id="cnvx1" class="img-fluid w-50"></canvas>

    </div>

    <!-- The buttons -->

    <div class="d-flex align-self-stretch gap-1">

      <button

        type="button"

        id="btn\_edge\_detection\_h"

        class="btn btn-primary flex-fill"

      >

        Sobel vertical edge detection

      </button>

      <button

        type="button"

        id="btn\_edge\_detection"

        class="btn btn-primary flex-fill"

      >

        Sobel horizontal edge detection

      </button>

      <button type="button" id="btn\_sharpen" class="btn btn-primary flex-fill">

        Sharpen

      </button>

      <button type="button" id="btn\_blur" class="btn btn-primary flex-fill">

        Blur

      </button>

      <button type="button" id="btn\_emboss" class="btn btn-primary flex-fill">

        Emboss

      </button>

    </div>

    <!-- Footer -->

    <footer class="display-6">Written by Mahmoud Darwish</footer>

    <!-- Boostrap JS -->

    <script

      src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/js/bootstrap.bundle.min.js"

      integrity="sha384-/bQdsTh/da6pkI1MST/rWKFNjaCP5gBSY4sEBT38Q/9RBh9AH40zEOg7Hlq2THRZ"

      crossorigin="anonymous"

    ></script>

    <!-- JQuery -->

    <script

      src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"

      integrity="sha512-894YE6QWD5I59HgZOGReFYm4dnWc1Qt5NtvYSaNcOP+u1T9qYdvdihz0PPSiiqn/+/3e7Jo4EaG7TubfWGUrMQ=="

      crossorigin="anonymous"

      referrerpolicy="no-referrer"

    ></script>

    <!-- Main Script -->

    <script>

      let canvasBefore = $("#cnvx0")[0];

      let canvasAfter = $("#cnvx1")[0];

      let contextBefore = canvasBefore.getContext("2d");

      let contextAfter = canvasAfter.getContext("2d");

      let img = new Image();

      let masks = {

        standard\_Sobel\_edge\_detector: [

          [-1, 0, 1],

          [-2, 0, 2],

          [-1, 0, 1],

        ],

        standard\_Sobel\_edge\_detector\_1: [

          [1, 2, 1],

          [0, 0, 0],

          [-1, -2, -1],

        ],

        sharpen: [

          [0, -1, 0],

          [-1, 5, -1],

          [0, -1, 0],

        ],

        blur: [

          [0.0625, 0.125, 0.0625],

          [0.125, 0.25, 0.125],

          [0.0625, 0.125, 0.0625],

        ],

        emboss: [

          [-2, -1, 0],

          [-1, 1, 1],

          [0, 1, 2],

        ],

      };

      // Loads an image into the provided canvas then excutes "thenDo"

      function loadImage(link, canvas, thenDo) {

        console.log(`Loading: ${link}`);

        let context = canvas.getContext("2d");

        img.src = link;

        img.crossOrigin = "Anonymous";

        img.onload = function () {

          // Use the intrinsic size of image in CSS pixels for the canvas element

          canvas.width = this.naturalWidth;

          canvas.height = this.naturalHeight;

          context.drawImage(this, 0, 0);

          console.log(`Loaded: ${link}`);

          thenDo();

        };

      }

      // returns an imageData object after grayscaling it's content

      let getGrayscaledImage = (canvas) => {

        let context = canvas.getContext("2d");

        let imageData = context.getImageData(0, 0, canvas.width, canvas.height);

        let data = imageData.data;

        for (let i = 0; i < data.length; i += 4) {

          let avg = 0;

          for (let j = i; j < i + 3; j++) {

            avg += data[j];

          }

          for (let j = i; j < i + 3; j++) {

            data[j] = avg / 3;

          }

          data[i + 3] = 255;

        }

        return imageData;

      };

      // draws to the provided canvas an imageData object and resized the canvas

      // to the size of the image provided

      let setDrawingContent = (image, canvas, imageData) => {

        canvas.width = image.naturalWidth;

        canvas.height = image.naturalHeight;

        let ctx = canvas.getContext("2d");

        ctx.putImageData(imageData, 0, 0);

      };

      let updateContents = (url) => {

        loadImage(url, canvasBefore, () => {

          setDrawingContent(img, canvasBefore, getGrayscaledImage(canvasBefore));

          setDrawingContent(img, canvasAfter, getGrayscaledImage(canvasBefore));

        });

      };

      // Converts the 1D array containting RGBA values to an array containing a single

      // value denoting the gray scale of each pixel

      function get1DGrayscaledDigitArray(data) {

        let newArr = [];

        for (let i = 0; i < data.length; i += 4) {

          let avg = 0;

          for (let j = i; j < i + 3; j++) {

            avg += data[j];

          }

          newArr.push(parseInt(avg / 3));

        }

        return newArr;

      }

      // draws a 1D array to the provided canvas by modifying the imageData object

      // accordingly

      function drawSinglePixelArray(grayScaleArray, canvas, imageData) {

        let data = imageData.data;

        let context = canvas.getContext("2d");

        for (let i = 0; i < grayScaleArray.length; i++) {

          for (let j = i \* 4; j < i \* 4 + 3; j++) {

            data[j] = grayScaleArray[i];

          }

          data[i \* 4 + 3] = 255;

        }

        context.clearRect(0, 0, canvas.width, canvas.height);

        context.putImageData(imageData, 0, 0);

      }

      // converts a 1D RGBA array to a matrix in order to make it easier to apply the mask

      function getMatrix(grayScaleArray, arrayWidth) {

        let matrix = [];

        let size = grayScaleArray.length;

        for (let i = 0; i < grayScaleArray.length / arrayWidth; i++) {

          matrix[i] = Array(arrayWidth);

        }

        for (let i = 0; i < matrix.length; i++) {

          for (let j = 0; j < matrix[i].length; j++) {

            matrix[i][j] = grayScaleArray[grayScaleArray.length - --size];

          }

        }

        return matrix;

      }

      // Applies a given mask to a matrix and returns the new

      function applyMaskToMatrix(matrix, mask) {

        let arrayWidth = matrix[0].length;

        let resultMatrix = [];

        for (let i = 0; i < matrix.length; i++) {

          resultMatrix.push(Array(arrayWidth));

          for (let j = 0; j < resultMatrix[i].length; j++) {

            resultMatrix[i][j] = 0;

          }

        }

        for (let i = 1; i < matrix.length - 1; i++) {

          for (let j = 1; j < matrix[i].length - 1; j++) {

            resultMatrix[i][j] =

              mask[0][0] \* matrix[i - 1][j - 1] +

              mask[0][1] \* matrix[i - 1][j] +

              mask[0][2] \* matrix[i - 1][j + 1] +

              mask[1][0] \* matrix[i][j - 1] +

              mask[1][1] \* matrix[i][j] +

              mask[1][2] \* matrix[i][j + 1] +

              mask[2][0] \* matrix[i + 1][j - 1] +

              mask[2][1] \* matrix[i + 1][j] +

              mask[2][2] \* matrix[i + 1][j + 1];

          }

        }

        return resultMatrix;

      }

      // Applies the provided mask to the loaded image and shows the result in the "after" canvas

      function applyMask(mask) {

        let img\_data = contextBefore.getImageData(

          0,

          0,

          canvasBefore.width,

          canvasBefore.height

        );

        // a 1-d array of the RGBA values listed as followes: [R,G,B,A,R,G,B,A.....]

        let data = img\_data.data;

        // Generate an array with each index representing a pixel

        let imgArray1D = get1DGrayscaledDigitArray(data);

        // Draw the new 1-d array represting a pixel grayscale value in each index

        let convolutedArray = getConvolutionResult(

          imgArray1D,

          canvasBefore.width,

          mask

        );

        drawSinglePixelArray(convolutedArray, canvasAfter, img\_data);

      }

      function getConvolutionResult(pixelsArray, arrayWidth, mask) {

        let matrix = getMatrix(pixelsArray, arrayWidth);

        let resultMatrix = applyMaskToMatrix(matrix, mask);

        let result1DPixelsArray = [];

        for (let i = 0; i < resultMatrix.length; i++) {

          for (let j = 0; j < resultMatrix[i].length; j++) {

            result1DPixelsArray.push(resultMatrix[i][j]);

          }

        }

        return result1DPixelsArray;

      }

      imageUrl.oninput = function () {

        updateContents(this.value);

      };

      $("#imageUrl").trigger("input");

      $("#btn\_edge\_detection").click(() => {

        applyMask(masks.standard\_Sobel\_edge\_detector\_1);

      });

      $("#btn\_edge\_detection\_h").click(() => {

        applyMask(masks.standard\_Sobel\_edge\_detector);

      });

      $("#btn\_sharpen").click(() => {

        applyMask(masks.sharpen);

      });

      $("#btn\_blur").click(() => {

        applyMask(masks.blur);

      });

      $("#btn\_emboss").click(() => {

        applyMask(masks.emboss);

      });

    </script>

  </body>

</html>