TIC4005 Project 1

Teng Kang Teng **A0211547L**

Table of Contents

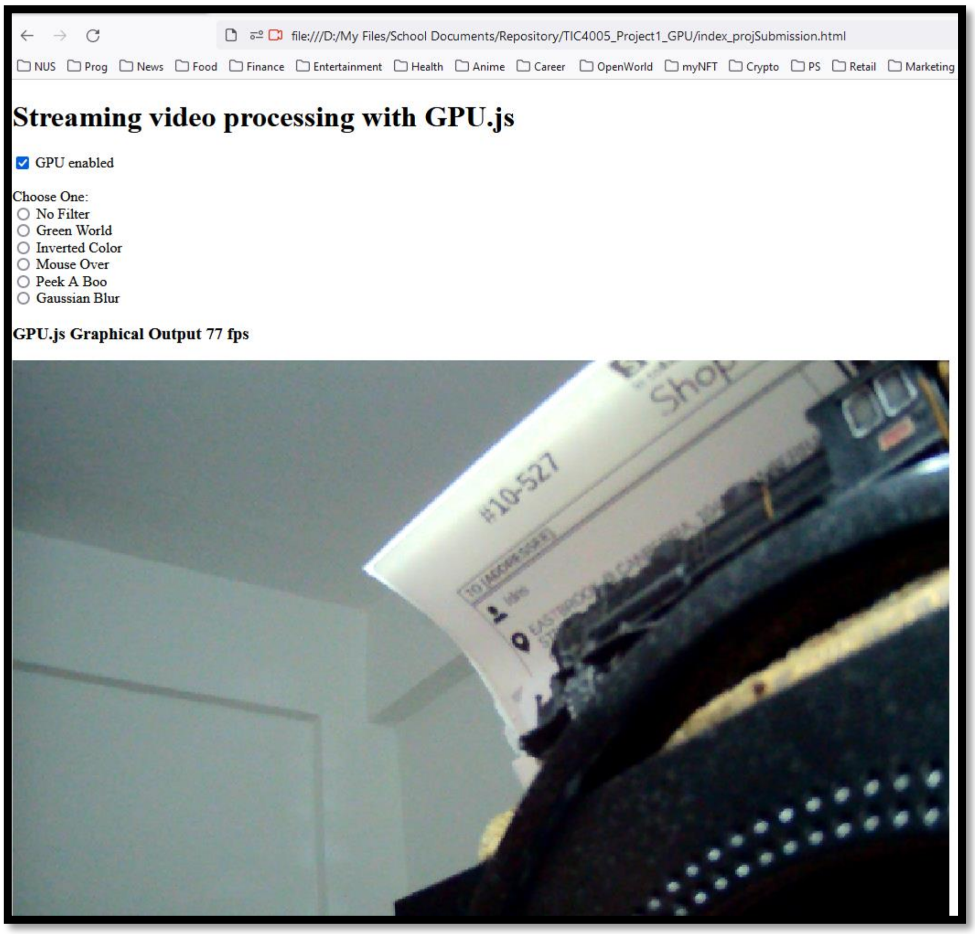
1 Project Outline

The project aims to develop several image processing functions which runs on either CPU or GPU. User can interact with the system by opening ***index_projSubmission.html*** using web browser. The project is developed using HTML and. The project demonstrates some of the image processing functions using GPU.js implementations.

The image processing features developed in this project are:

- Green World
- Inverted Color
- Mouse over
- Peekaboo
- Gaussian Blur

2 Front End User Interface


file:///D:/My Files/School Documents/Repository/TIC4005_Project1_GPU/index_projSubmission.html
NUS
Prog
News
Food
Finance
Entertainment
Health
Anime
Career
OpenWorld
myNFT
Crypto
PS
Retail
Marketing

Streaming video processing with GPU.js

[✓] GPU enabled

Choose One:
- ○ No Filter
- ○ Green World
- ○ Inverted Color
- ○ Mouse Over
- ○ Peek A Boo
- ○ Gaussian Blur

GPU.js Graphical Output 77 fps

Front End User Interface

User can toggle between GPU enabled to control if the video computation is processed through CPU or GPU.

Also, user can choose one of the filters to apply to the image.

3 Image Processing Feature

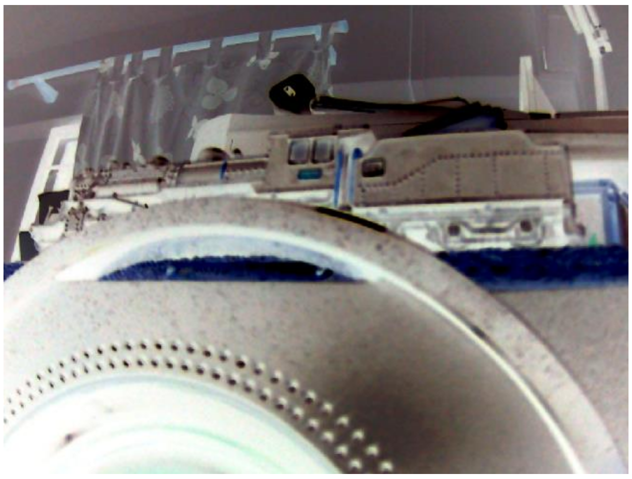
By toggling the radio buttons, user can control which filter to activate. The radio button will trigger ***onclick*** function to change the ***filterValue***, which in turn changes how the pixel is processed. Different ways of handling the pixel can generate different value, which will be translated into the different image output.

3.1 Green World

```
 function (frame, filter, filterValue, mX, mY, k) {
      const pixel = frame[this.thread.y][this.thread.x];

      var r = pixel[0];
      var g = pixel[1];
      var b = pixel[2];
      var a = pixel[3];
      var result = [r, g, b, a];

      if (filterValue == 1) {
        result = greenWorld(r, g, b, a);

      }
   …
}

function greenWorld(r, g, b, a) {
  return [0, g, 0, a];
}
```

By changing the red and blue value to 0, these two colors are eliminated from the ***pixel***. The resulting ***frame*** would only be green, thus achieving a green world effect.

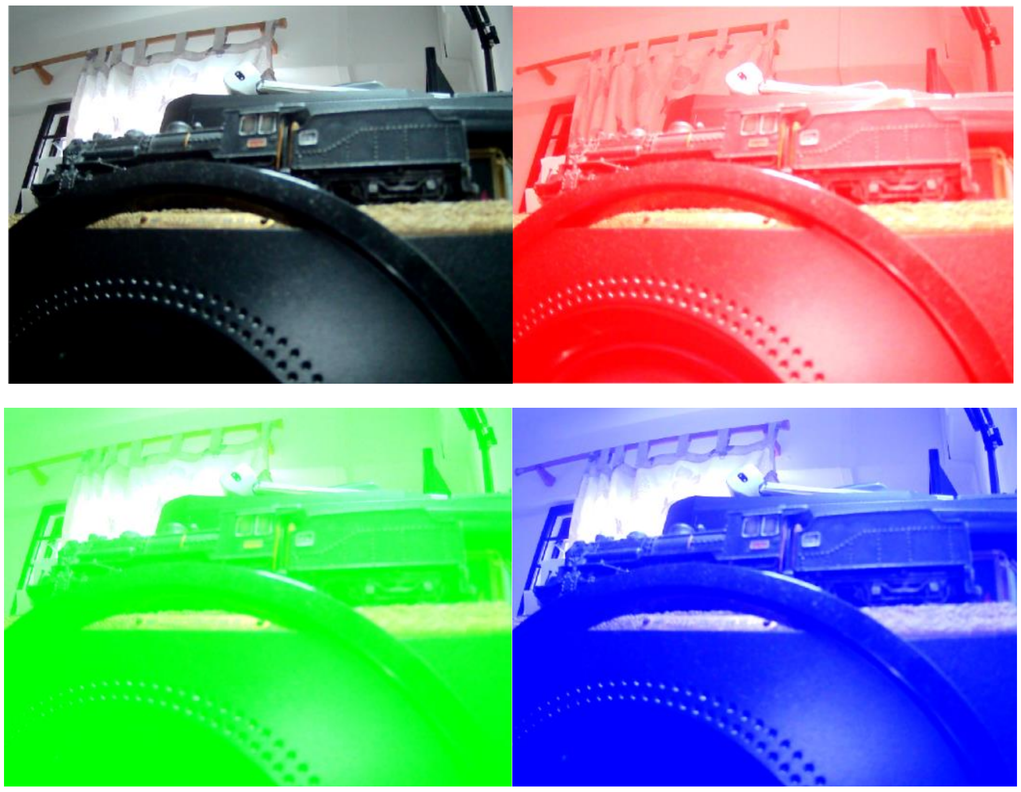
3.2 Inverted Color



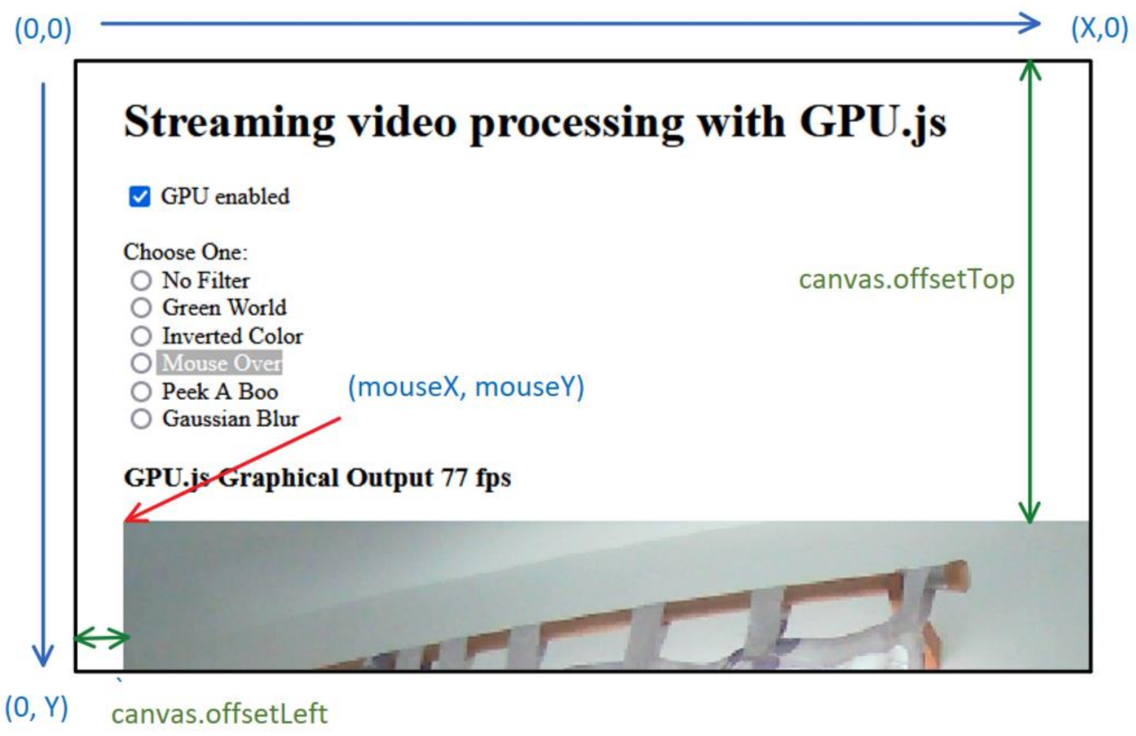
```
function invertedColor1(r, g, b, a) {
  return [1 - r, 1 - g, 1 - b, a];
}
```

By inverting all the value by doing ***1- [the color]***, pixel color is inverted. The resulting ***frame*** would have inverted color, black would turn to white, red would turn to green, and so on. Thus this achieves an inverted color effect.

3.3 Mouse Over



```
// THIS IS THE IMPORTANT STUFF
  const kernel = gpu.createKernel(
    function (frame, filter, filterValue, mX, mY, k) {
      const pixel = frame[this.thread.y][this.thread.x];

      …

  canvasParent.appendChild(kernel.canvas);
  const videoElement = document.querySelector('video');

  // initialize the kernel
  kernel(videoElement, filter.checked, filterValue, 0, 0, k_value);

  // initialize the kernel and mouse position
  const canvas = kernel.canvas;
  var mouseX = 1024 / 2;
  var mouseY = 768 / 2;

  // compute mouse position in the canvas when cursor move in the video screen
  canvas.addEventListener("mousemove", setMousePosition, false);

  function setMousePosition(e) {
    console.log("(Before) mouse Y: " + mouseY);
    console.log("(Before) e.clientY: " + e.clientY);
    console.log("(Before) canvas.offsetTop: " + canvas.offsetTop);
    mouseX = e.clientX - canvas.offsetLeft;
    mouseY = 768 - (e.clientY - canvas.offsetTop);
    console.log("mouseX: " + mouseX + "  mouse Y: " + mouseY);
    kernel(videoElement, filter.checked, filterValue, mouseX, mouseY, k_value);
  }
  …
  };
}

…

function mouseOver(r, g, b, a, mX) {
  if (mX <= 256) {
    return [r, g, b, a];
  } else if (256 < mX && mX <= 512) {
    return [255, g, b, a];
  } else if (512 < mX && mX <= 768) {
    return [r, 255, b, a];
  } else {
    return [r, g, 255, a];
  }
}
```



Streaming video processing with GPU.js

☑ GPU enabled

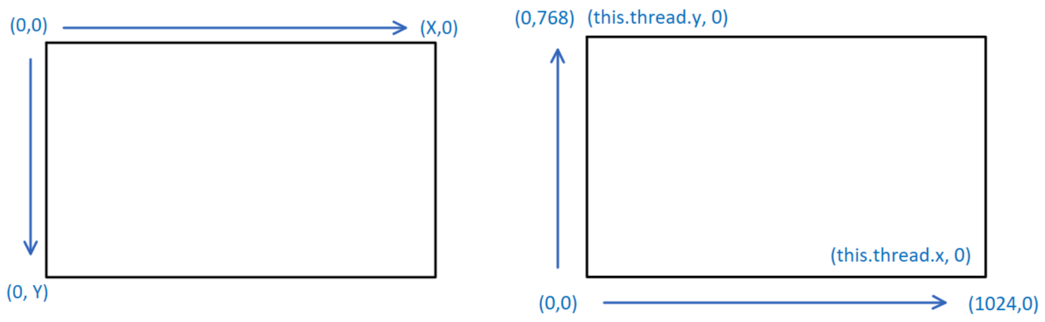
Choose One:
- ○ No Filter
- ○ Green World
- ○ Inverted Color
- ○ Mouse Over
- ○ Peek A Boo
- ○ Gaussian Blur

GPU.js Graphical Output 77 fps

Mouse position relative to the canvas is captured by performing calculation. For ***E.clientX***, ***e*** is the ***MouseEvent***, ***clientX*** is a read-only property of the ***MouseEvent*** interface. It provides the horizontal coordinate of the mouse within the application's viewport at which the event occurred, likewise for ***clientY*** which provides the vertical coordinate.

Canvas.offsetLeft returns the difference between the left edge of the element and the window, likewise ***Canvas.offsetTop*** returns the difference between the top edge of the element and the window.

By performing the ***e.clientX – canvas.offsetLeft*** and ***e.clientY – canvas.offsetTop***, origin of the mouseX & mouseY is set to the top left edge of the video canvas.

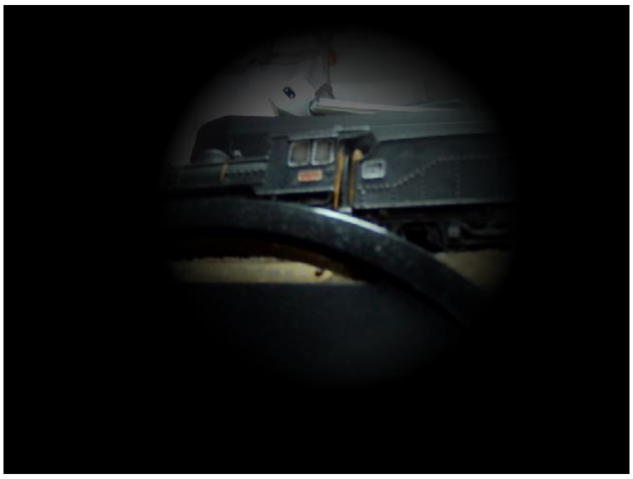


Coordinate System of the canvas (Left) and GPU.js (right)

The coordinate system of the viewport (***canvas*** in the HTML) begins from the top-left hand corner, whereas the coordinate system of the GPU.js begins from the bottom-left hand corner. To compensate for the difference, and also to set the origin of mouseX and mouseY to be at the bottom-left hand corner, ***mouseY = 768 - (e.clientY - canvas.offsetTop);***

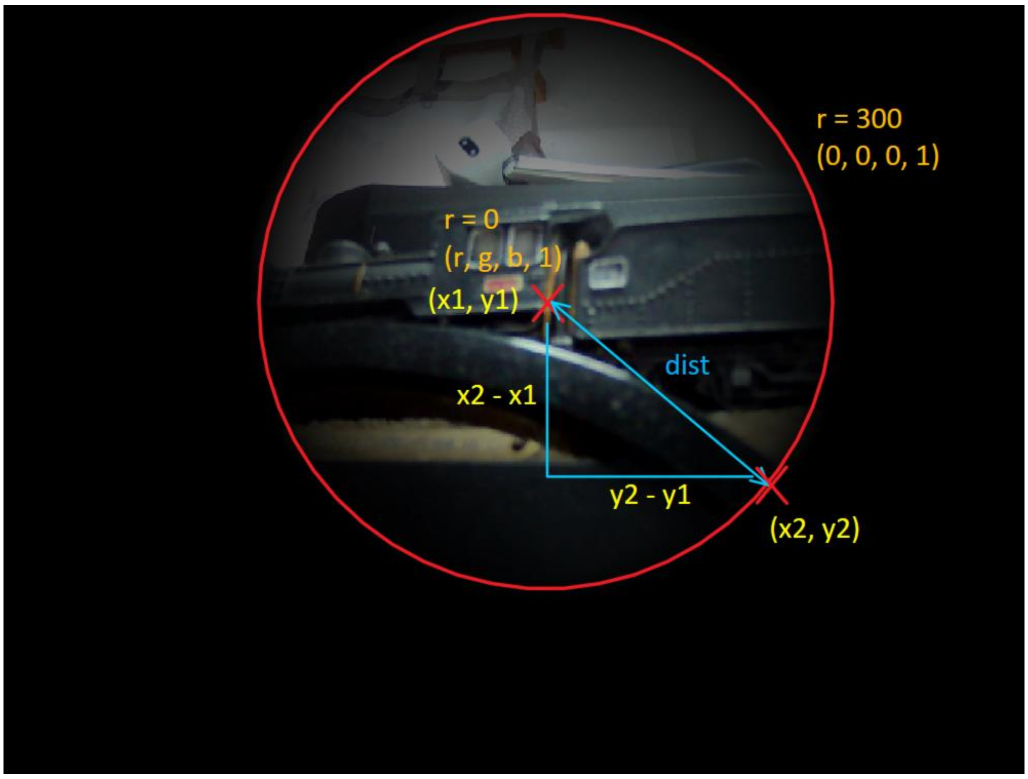
After the mouse position has been set, as the mouse is moved from left to right, depending on the ***mX*** value, ***rbga*** values to adjusted accordingly, thus giving the mouse over effect.

3.4 Peekaboo



```
function peekaboo(r, g, b, a, mX, mY) {
  var dist = calcDistance(mX, mY, this.thread.x, this.thread.y);
  var factor = calcFactor(dist, 300, 0);
  return [r * factor, g * factor, b * factor, a]
}

function calcDistance(x1, y1, x2, y2) {
  // return distance between two points using pythagoras theorem
  var a = x2 - x1;
  var b = y2 - y1;
  return Math.sqrt(a * a + b * b);
}

function calcFactor(dist, maxRange, minRange) {
  // return normalized value between range as a factor between 1 to 0
  if (dist <= minRange) {
    return 1;
  }
  if (dist >= maxRange) {
    return 0;
  }
  return (maxRange - dist - minRange) / (maxRange - minRange);
}
```

Peekaboo is an effect which the area around the mouse curser is lighten, and all the other area are darken. Brightness of a pixel can be controlled by ***[rgba] * factor***. The higher the factor, the brighter the pixel; the lower the factor, the darker the pixel.



The first step is to calculate the distance of the pixel away from the mouse cursor. Using Pythagoras Theorem, distance is calculated using ***$\sqrt{(x2-x1)^2 + (y2-y1)^2}$*.** The second step is to calculate the factor. The pixel exactly on the mouse cursor has the factor of 1, which return the original ***rbga*** value from the videoframe. As the distance increases, the factor decreases from 1, up to the lowest factor of 0. By performing ***[rgba] * factor***, the pixel further away from cursor centre is darken, thus achieving the peekaboo effect.

3.5 Gaussian Blur

```
let k_value =
  [1, 4, 6, 4, 1,
    4, 16, 24, 16, 4,
    6, 24, 36, 24, 6,
    4, 16, 24, 16, 4,
    1, 4, 6, 4, 1];

…

if (this.thread.y > 0 + 2 && this.thread.y < 768 - 2 && this.thread.x < 1024 - 2 &&
this.thread.x > 0 + 2) {
    const r1c1 = frame[this.thread.y + 2][this.thread.x - 2];
    const r1c2 = frame[this.thread.y + 2][this.thread.x - 1];
    const r1c3 = frame[this.thread.y + 2][this.thread.x + 0];
    const r1c4 = frame[this.thread.y + 2][this.thread.x + 1];
    const r1c5 = frame[this.thread.y + 2][this.thread.x + 2];

    const r2c1 = frame[this.thread.y + 1][this.thread.x - 2];
    const r2c2 = frame[this.thread.y + 1][this.thread.x - 1];
    const r2c3 = frame[this.thread.y + 1][this.thread.x + 0];
    const r2c4 = frame[this.thread.y + 1][this.thread.x + 1];
    const r2c5 = frame[this.thread.y + 1][this.thread.x + 2];

    const r3c1 = frame[this.thread.y][this.thread.x - 2];
    const r3c2 = frame[this.thread.y][this.thread.x - 1];
    const r3c3 = frame[this.thread.y][this.thread.x + 0];
    const r3c4 = frame[this.thread.y][this.thread.x + 1];
    const r3c5 = frame[this.thread.y][this.thread.x + 2];

    const r4c1 = frame[this.thread.y - 1][this.thread.x - 2];
    const r4c2 = frame[this.thread.y - 1][this.thread.x - 1];
    const r4c3 = frame[this.thread.y - 1][this.thread.x + 0];
    const r4c4 = frame[this.thread.y - 1][this.thread.x + 1];
    const r4c5 = frame[this.thread.y - 1][this.thread.x + 2];

    const r5c1 = frame[this.thread.y - 2][this.thread.x - 2];
    const r5c2 = frame[this.thread.y - 2][this.thread.x - 1];
    const r5c3 = frame[this.thread.y - 2][this.thread.x + 0];
    const r5c4 = frame[this.thread.y - 2][this.thread.x + 1];
    const r5c5 = frame[this.thread.y - 2][this.thread.x + 2];
    var divisor = 0;
    var col = [0, 0, 0];

    for (var j = 0; j < 15; j++) {
        divisor = divisor + k[j];
    }

    for (var i = 0; i < 3; i++) {

        col[i] =
            r1c1[i] * k[0] + r1c2[i] * k[1] + r1c3[i] * k[2] + r1c4[i] * k[3] + r1c5[i] * k[4]
            + r2c1[i] * k[0] + r2c2[i] * k[1] + r2c3[i] * k[2]
                + r2c4[i] * k[3] + r2c5[i] * k[4]
            + r3c1[i] * k[0] + r3c2[i] * k[1] + r3c3[i] * k[2]
                + r3c4[i] * k[3] + r3c5[i] * k[4]
            + r4c1[i] * k[0] + r4c2[i] * k[1] + r4c3[i] * k[2]
                + r4c4[i] * k[3] + r4c5[i] * k[4]
            + r5c1[i] * k[0] + r5c2[i] * k[1] + r5c3[i] * k[2]
                + r5c4[i] * k[3] + r5c5[i] * k[4];

        col[i] = col[i] / divisor * 2.2;
        result = [col[0], col[1], col[2], 1];
    }
}
```

Gaussian blur is a kind of image blurring technique that is built using the convolution matrix. If the image has low light that result in a lot of noise, Gaussian blur can mute the noise, soften the image, and allows the text to stand out clearer.

The Gaussian matrix used in this project is a 5 by 5 matrix ***k***. The center part of the matrix has the highest value, decreasing symmetrically as the value distanced away from the center.

For each pixel, each value of the pixel is multiplied with the respective value at the same position of k. For example. Assuming the red at row 1 column 1 of pixel has value of 100. Red at Row 1 Column 1 of pixel is multiplied with value at ***k[0]***, giving ***r1c1[0] * k[0] = 100 * 1 = 100***. The multiplication goes on for the rest of the pixels and matrix. The sum of these numbers is divided by the summation of the matrix values (***divisor***) to determine the pixel value. As the resulting image appears darker than original, it's brightened by performing ***2.2**. Thus, by performing ***col[i] = col[i] / divisor * 2.2*** following convolution, the resulting image is blurred.

4 Limitation

The project is developed starting from no knowledge of JavaScript. I have read up and learn JavaScript for this project. Also, understanding how the kernel and GPU.js works took a significant amount of time. As such, the number of features developed in this project is low and they are not very sophisticated.

5 Project Links

The development of this project can be found at:

https://github.com/EndireKT/TIC4005_Project1_GPU

6 References

The project is developed by learning and following a wide range of video, tutorial, and research papers. The referenced online contents can be found below:

GPU.js - GPU accelerated Javascript. GPU.js - GPU accelerated JavaScript. (n.d.). Retrieved October 3, 2022, from https://gpu.rocks/#/

Gpujs. (n.d.). *Quick concepts · GPUJS/gpu.js wiki*. GitHub. Retrieved October 3, 2022, from https://github.com/gpujs/gpu.js/wiki/Quick-Concepts

shiffman. (2015, July 24). *10.5: Image processing with pixels - processing tutorial*. YouTube. Retrieved October 3, 2022, from https://www.youtube.com/watch?v=j-ZLDEnhT3Q

Chinnathambi, K. (n.d.). *Follow the mouse cursor*. kirupa.com. Retrieved October 3, 2022, from https://www.kirupa.com/canvas/follow_mouse_cursor.htm

Computerphile. (2015, November 4). *Finding the edges (Sobel operator) - computerphile*. YouTube. Retrieved October 3, 2022, from https://www.youtube.com/watch?v=uihBwtPIBxM

YouTube. (2017, November 9). *WebGL + GPU = amazing results!* YouTube. Retrieved October 3, 2022, from https://www.youtube.com/watch?v=qkDg-Y9iHBA

Gaussian blur. Gaussian Blur - an overview | ScienceDirect Topics. (n.d.). Retrieved October 3, 2022, from https://www.sciencedirect.com/topics/engineering/gaussian-blur

MouseEvent.clientX - web apis: MDN. Web APIs | MDN. (n.d.). Retrieved October 3, 2022, from https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent/clientX