**Rotation Task**

Contents

[1. Purpose of the Document 1](#_Toc61311443)

[2. Pre-Requisites 1](#_Toc61311444)

[3. Steps to run the html 2](#_Toc61311445)

[4. The function --> rotate(image: ImageData, angle: double) : ImageData 4](#_Toc61311446)

[5. Radians Calculation and Tilting angle calculation. 7](#_Toc61311447)

[6. Rotate2 button: uses function rotateImageWithCanvasTilt() 8](#_Toc61311448)

[7. Other Approaches: Matrix approach 9](#_Toc61311449)

[8. References: 10](#_Toc61311450)

# 1. Purpose of the Document

To submit the explanation of rotation task algorithm, that converts the provided RGBA data buffer that is an ImageData object, and returns a valid RGBA data buffer - an ImageData object, when the image is rotated to a particular angle.

+ve Angle is a clockwise rotation

-ve Angle is an Anti-clockwise rotation

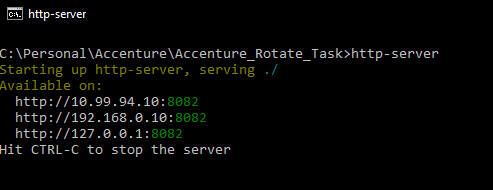
The proposed function should follow the input and output signature as follows  
 **rotate(image: ImageData, angle: double) : ImageData**

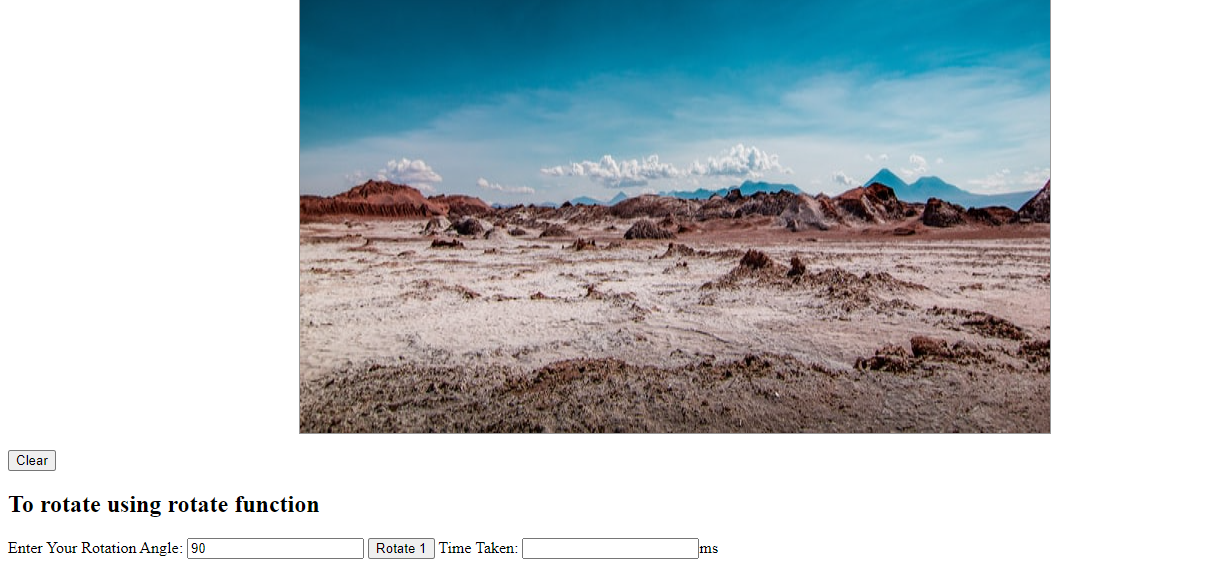
# 2. Pre-Requisites

A. rotator.html  
B. MyImage.jpeg (Any RGBA image of some format (jpeg))  
C. rotate function inside rotator object with method signature **rotate(image: ImageData, angle: double) : ImageData**D. Install http-server in node globally using the command  
**npm install -g http-server**

**(Since we run this rotator.html with Canvas element, to have a visual representation of the rotation. And to avoid the cross-origin).**

# 3. Steps to run the html

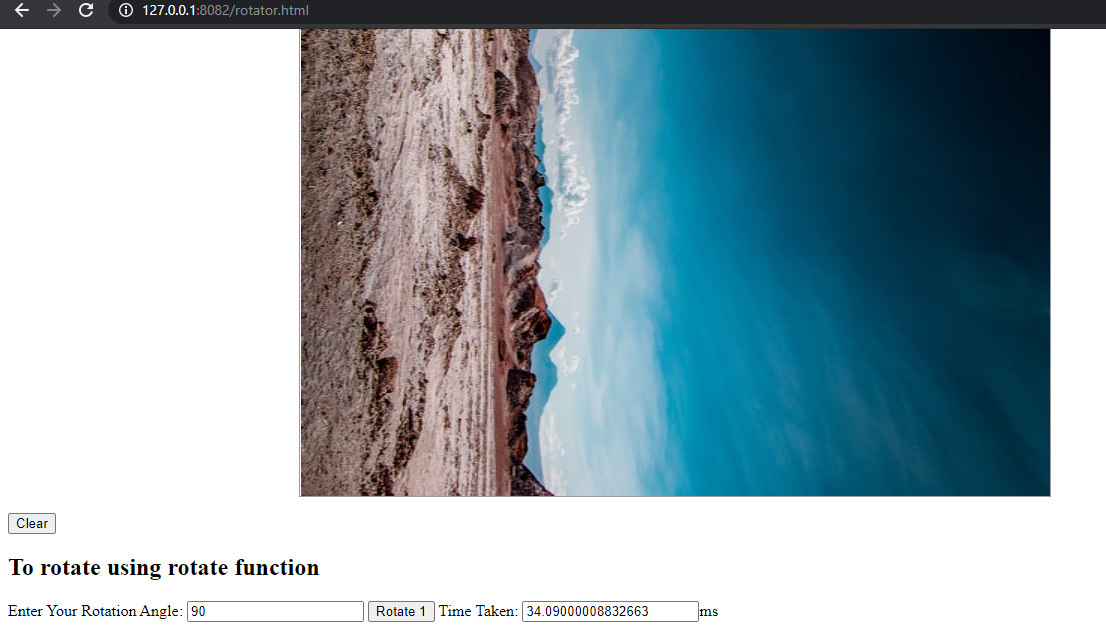
1. Place the rotator.html and MyImage.jpeg in any physical folder.  
2. Start the http server in the particular folder using the command **http-server  
(Reference: https://www.npmjs.com/package/http-server)  
**

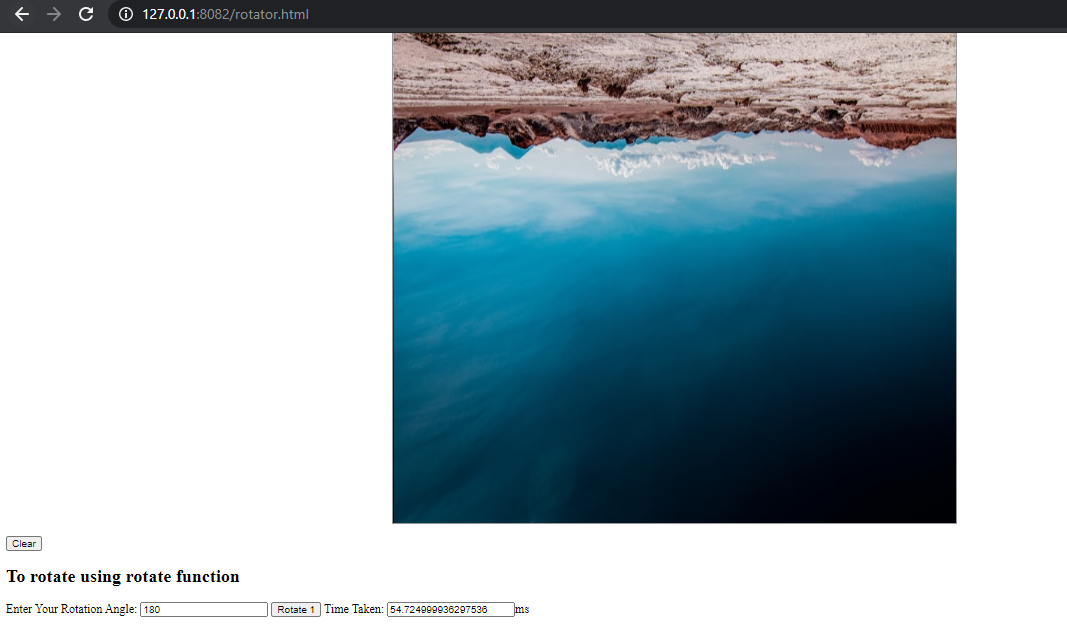
3.You can go to the browser and see the list of available files in **http://127.0.0.1:8082  
(Localhost)  
  
4. Click on rotator.html to access the UI html  
**

**Form info:  
"Enter Your Rotation Angle:"** User desired angle in double

**"Rotate 1"** button: To call the function **rotate(image: ImageData, angle: double) : ImageData ,** and return the modified data buffer. (This will be transformed into the canvas display to best possible fit)

**Time Taken:** Total time taken to run the function **rotate(image: ImageData, angle: double) : ImageData** in Milliseconds using **performance.now()**

5.After rotating to a particular angle:  
90 degree: 

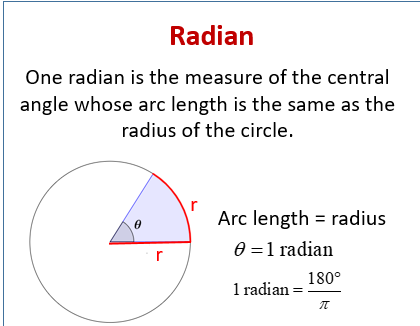
**180 degree:** 

# 4. The function --> rotate(image: ImageData, angle: double) : ImageData

|  |
| --- |
| **function rotate(imageData, userAngle) {**  **var t0 = performance.now();**  **if (imageData == null || imageData == undefined || imageData.data == null || imageData.data == undefined) {**  **alert("Please try with a valid image data of RGBA type");**  **return;**  **}**  **if (imageData.data.length % 4 != 0) {**  **alert("Image data of RGBA type is of invalid length");**  **return;**  **}**  **if (userAngle == null || userAngle == undefined || userAngle == '' || userAngle == '0' || userAngle == '360' ) {**  **alert("Please provide a valid angle to rotate the image.");**  //this.document.getElementById("angle2").value=0;  **return;**  **}**  **if (parseInt(userAngle) > 360) {**  **userAngle = userAngle % 360;**  **console.log("final positive userAngle is:" + userAngle);**  **}**  **if (parseInt(userAngle) < -360) {**  **userAngle = userAngle % 360;**  **console.log("final negative userAngle is:" + userAngle);**  **}**    /\*\*  Angle in radians = Angle in degrees x PI / 180.  \*\*/  **var piVal = Math.PI;**  **radianAngle = userAngle \* (piVal / 180);**  **if (radianAngle == '' || radianAngle == undefined) {**  **radianAngle = 90 \* (piVal / 180);**  **}**  **console.log("Angle in radian is" + radianAngle);**    **let arr = imgData.data;**  **let updatedArr = new Uint8ClampedArray(arr.length);**  **var center\_x = imgData.width / 2;**  **var center\_y = imgData.height / 2;**    **console.log("center\_x" + center\_x);**  **console.log("center\_y" + center\_y)**  **var outputCounterForUpdate = 0;**  **var myWidth = imgData.width;**  **var myHeight = imgData.height;**  **for (var x = 0; x < myWidth; x++) {**  **for (var y = 0; y < myHeight; y++) {**  /\*\*  In linear algebra, a rotation matrix is a transformation matrix that is used to perform a rotation in Euclidean space. For example, using the convention below, the matrix rotates points in the xy-plane counterclockwise through an angle θ with respect to the x axis about the origin of a two-dimensional Cartesian coordinate system.    X2 = cos(θ)\*x1 + sin(θ)\*y1;  y2 = sin(θ)\*x1 + cos(θ)\*y1;    To perform the rotation on a plane point with standard coordinates v = (x0,y0), it should be written as a column vector, and multiplied by the matrix R:    X2 = cos(θ)\*(x1-x0) + sin(θ)\*(y1-y0);  y2 = sin(θ)\*(x1-x0) + cos(θ)\*(y1-y0);    //var newxp = parseInt((x - center\_x) \* Math.cos(radianAngle) + (y - center\_y) \* Math.sin(radianAngle));  //var newyp = parseInt((x - center\_x) \* Math.sin(radianAngle) - (y - center\_y) \* Math.cos(radianAngle));      //To perform the same by finding the position of the current rotated image's pixel value from actual //input data  X2 = sin(θ)\*(x1-x0) + cos(θ)\*(y1-y0) +y0;  y2 = cos(θ)\*(x1-x0) - sin(θ)\*(y1-y0)+x0;  \*\*/  **var newxp = parseInt((x - center\_x) \* Math.sin(radianAngle) + (y - center\_y) \* Math.cos(radianAngle) + center\_y);**  **var newyp = parseInt((x - center\_x) \* Math.cos(radianAngle) - (y - center\_y) \* Math.sin(radianAngle) + center\_x);**  /\*\*  // 1.Think always image as an X x Y space, and two find a particular pixel position, loop through all the rows until the height of the image i,e (newyp \* myWidth), and in the last rows go till the particular pixel i.e add till (newxp)  // 2. Since we have four colour cooridnates for each pixel(R,G,B,A), multiply the result by 4 to find the starting point of the required pixel in dataBauffer.  \*\*/  **var newIndex = 4 \* (newxp + newyp \* myWidth);**  **updatedArr[outputCounterForUpdate] = arr[newIndex];** // setting the Red value of identified pixel from input image into goint to be rotated image  **updatedArr[outputCounterForUpdate + 1] = arr[newIndex + 1];** // setting the Green value of identified pixel from input image into goint to be rotated image  **updatedArr[outputCounterForUpdate + 2] = arr[newIndex + 2];** // setting the Blue value of identified pixel from input image into goint to be rotated image  **updatedArr[outputCounterForUpdate + 3] = arr[newIndex + 3];** // setting the Alpha value of identified pixel from input image into goint to be rotated image    **outputCounterForUpdate = outputCounterForUpdate + 4; // to go to the next pixel**  **}**  **}**  // Creating a new ImageData Object which will be returned in this function.  **var imageDataUpdatedAsImageDataObj = new ImageData(updatedArr, myWidth, myHeight);**  **console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");**  **console.log(imageDataUpdatedAsImageDataObj);**  **console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");**  **var t1 = performance.now();**  **console.log("Call to rotate() took " + (t1 - t0) + " milliseconds.");**  //this.document.getElementById("timeTaken2").value=(t1 - t0);  **return imageDataUpdatedAsImageDataObj;**  **}** |

# 5. Radians Calculation and Tilting angle calculation.

1. Radians:

One radian is the angle in which the arc length is equivalent to the radius of the circle. 

So the user provided angle will be converted into radians using the formula

**1 radian = angle \* (π/180)**

2. To calculate the pixel position after rotation

A. The coordinates of a point (x1, y1) when rotated by an angle θ around (x0, y0) become (x2, y2), as shown by the following equation:

X2 = cos(θ)\*(x1-x0) + sin(θ)\*(y1-y0);

y2 = sin(θ)\*(x1-x0) + cos(θ)\*(y1-y0);

B. By rotating the image about the origin (0, 0)

X2 = cos(θ)\*x1 + sin(θ)\*y1;

y2 = sin(θ)\*x1 + cos(θ)\*y1;

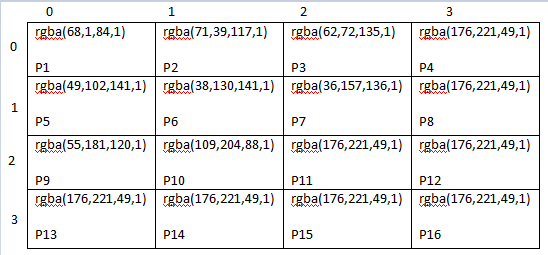
C. **To calculate the corresponding position in input image, for an output rotated image's coordinates of a point for a particular radian(angle) (Note: This is reverse to the logic of A and B)**

X2 = sin(θ)\*(x1-x0) + cos(θ)\*(y1-y0) +y0;

y2 = cos(θ)\*(x1-x0) - sin(θ)\*(y1-y0)+x0;

This rotate function (**rotate(image: ImageData, angle: double) : ImageData**) will use **"C"** to get the pixel info from input image and generate the arraybuffer of the output image.

3. Pixel location finding : **var newIndex = 4 \* (newxp + newyp \* myWidth);**



To find the position of P11, follow these steps  
(P11 sits in 3rd row and 3rd column)

1. Multiply the widht by height in completed pixels (P1 to P8).   
so 4\* 2 = 8 pixels,

2. Add the remaining pixels in third row, until P11, which is p9 and P10. i.,e two pixels

So **newxp + (newyp \* myWidth)**  is equals to 2 + 8 = P10.

This is multiplied by 4, since each pixel represents rgba in the imageData.data

So, the index will be 4(2 + 8) = 40, which means 41 to 44 items in the imageDataBuffer is the corresponding colour data we are looking for.

# 6. Rotate2 button: uses function rotateImageWithCanvasTilt()

# 

This Rotate 2 button uses the canvas Transform to find the position of the context ,and transform the context and draw the image and get the imageData out of it.

context.setTransform(a,b,c,d,e,f);

a Horizontal scaling  
b Horizontal skewing  
c Vertical skewing  
d Vertical scaling  
e Horizontal moving  
f Vertical moving

|  |
| --- |
| **var scale1 = Math.cos(ang1) \* dist / image.height;**  **var scale2 = Math.cos(ang2) \* dist / image.width;**  **var scale = Math.max(scale1, scale2);**  **var dx = Math.cos(angle) \* scale; // Horizontal scaling**  **var dy = Math.sin(angle) \* scale; // Horizontal skewing**  **ctx.setTransform(dx, dy, -dy, dx, canvas.width / 2, canvas.height / 2); ;**  **ctx.drawImage(image, -image.width / 2, -image.height / 2, image.width, image.height);** |

# 7. Other Approaches: Matrix approach

The imageData.data buffer which is multiples of 4, can be converted in to matrix of width\*height of array of 4. Then this array of elements is rearranged using columns reversal and rows reversal for Angles 90° or 180° or 270°.

Commented function: var **rotateMatrix = function (matrix, direction)**

# 8. References:

1. https://www.sciencedirect.com/topics/computer-science/image-rotation

2. https://www.onlinemathlearning.com/degrees-radians.html

3. https://www.codingame.com/playgrounds/2524/basic-image-manipulation/transformation