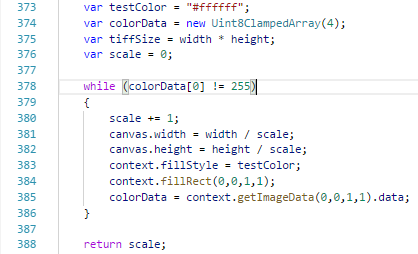
**RGBA**

**Problem:** An image is too large to display on a given device.

**Solution:** Scale the image down to a size the given device supports by averaging groups of pixels (of at least 4) and mapping each group to 1 pixel in the filtered image.

**Detailed Solution:**

*Finding the needed Scale:*



Determining a supported canvas size can be done by creating the canvas of the desired size, drawing a pixel to the canvas, and then reading the pixel back. If the pixel cannot be written, then the data read back will be an array of all zeros. If the pixel can be written the values of the array will match the color the pixel set to.

In the case where the pixel cannot be set, increment the scale by 1 and try again with the new desired canvas size. Repeat until the pixel can be successfully read.

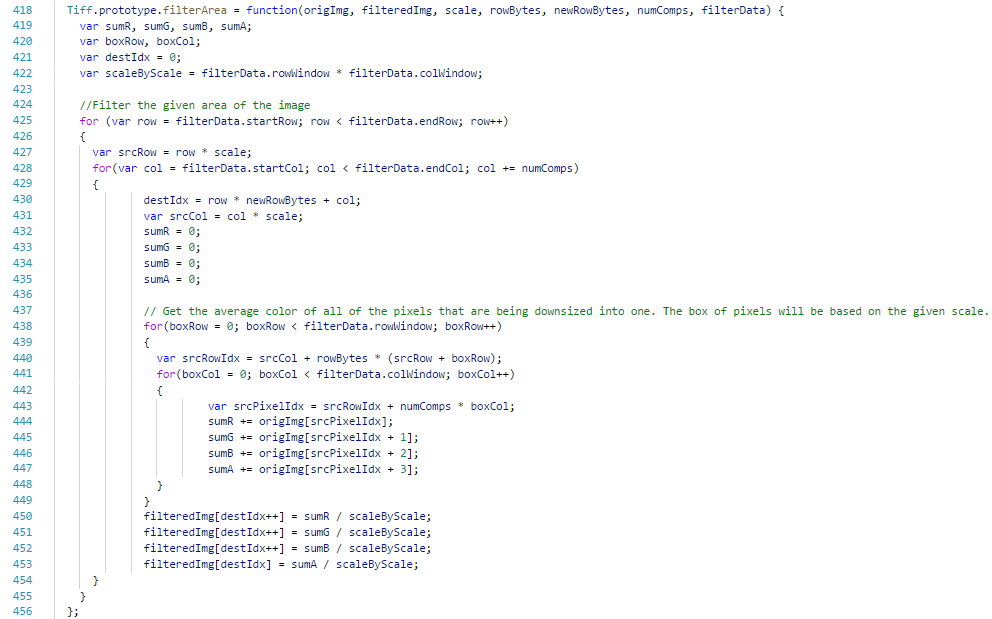
*Calculating the required variables:*

|  |  |  |
| --- | --- | --- |
| **Variable** | **Description** | **Calculation** |
| **drawWidth** | The number of pixels that will make up 1 row in the new filtered image. | The original image’s width divided by the scale rounded upwards. |
| **drawHeight** | The number of pixels that will make up 1 column in the new filtered image. | The original image’s height divided by the scale rounded upwards. |
| **numComps** | The number of components in each pixel. | For Tiff.js every tiff is coming in as an RGBA image. Red, Green, Blue, and Alpha are each separate components. Each component takes up 1 byte of memory. Each pixel takes 4 bytes. |
| **rowBytes** | The number of bytes used by each row of pixels in the original unaltered image. | The original image’s width multiplied by the number of components for each pixel. |
| **newRowBytes** | The number of bytes used by each row of pixels in the new filtered image. | The new image’s width multiplied by the number of components for each pixel. |
| **newImgArray** | The Uint8Array with enough bytes for the new image. | The size of the array is the area in bytes of the new image. This is the new image’s number of rows multiplied by the new image’s bytes per row. |

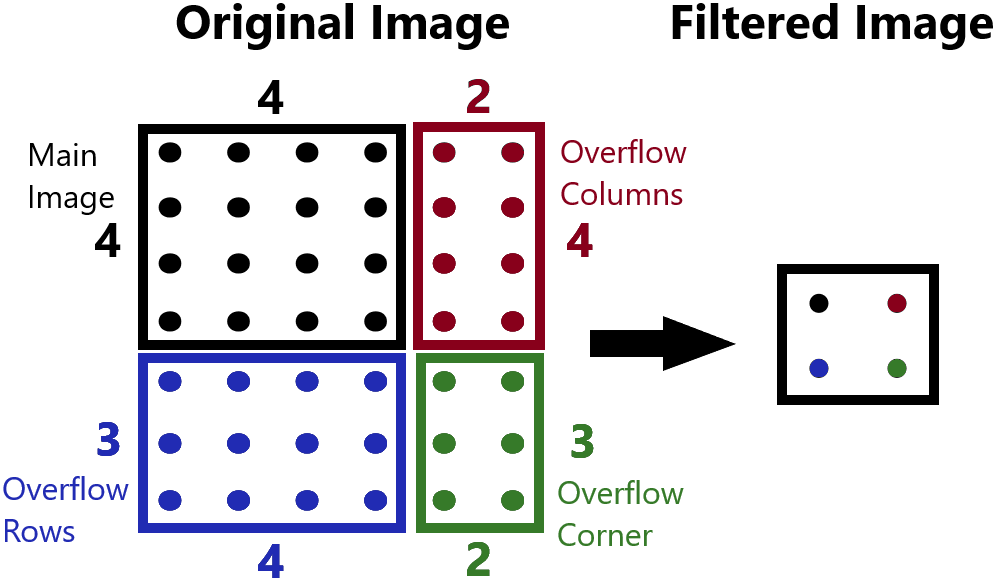
*Determining the Overflow Areas:*

|  |  |  |
| --- | --- | --- |
| **Variable** | **Description** | **Calculation** |
| **overflowRowCnt, overflowColCnt** | The remainder of rows and columns after dividing the rest of the rows and columns in the original unaltered image evenly by the scale. | The original height/width modded by the scale. |
| **overflowDestRow** | The index in the new filtered image that maps to the pixel calculated from the overflow rows. | If there are no overflow rows, then this will refer height of the new image. If there are overflow rows, then this will refer to the index of the last row in the new image. This is the new image’s height - 1. |
| **overflowDestCol** | The index in the new filtered image that maps to the pixel calculated from the overflow columns. | If there are no overflow columns, then this will be set to the new image’s number of bytes per row. If there are overflow columns, then this will be set to the index of the last column in the new image. This is the new image’s number of bytes per row minus 1 pixel worth of bytes. |
| **startRow** | The first row in the filtered image to begin mapping pixels to. | 0 if the rows being filtered are in the Main Image.  **overflowDestRow** if the rows being filtered are in the overflow rows area. |
| **endRow** | The row index in the filtered image to stop mapping pixels. | **overflowDestRow** if filtering the main image.  **drawHeight** if filtering an area with overflow rows. |
| **startCol** | The first column in the filtered image to begin mapping pixels to. | 0 if the columns being filtered are in the Main Image.  **overflowDestCol** if the columns being filtered are in the overflow columns area. |
| **endCol** | The column index in the filtered image to stop mapping pixels. | **overflowDestCol** if filtering the main image.  **newRowBytes** if filtering an area with overflow columns. |
| **rowWindow** | The number of rows in the original image that will be mapped to 1 row in the filtered image. | **scale** for filtering the main image.  **overflowRowCnt** if filtering an area with overflow rows. |
| **colWindow** | The number of columns in the original image that will be mapped to 1 column in the filtered image. | **scale** for filtering the main image.  **overflowColCnt** if filtering an area with overflow rows. |

*Filtering the image:*



For an image that is completely divisible by **scale** the above method will only need to be called once on the image, with the provided values set appropriately. However, with the scenario below:

*Scale*: 4

*OverflowRowCnt*: 3

*OverflowColCnt*: 2

This image would need to go through **filterArea(…)** 4 times. Once for the main image, once for the overflow rows area, once for the overflow column area, and once for the overflow corner area.

The **filterArea(…)** algorithm loops through the rows and columns in the filtered image. It uses this row or column value to calculate the **srcRow** or **srcCol** of the original image where we will begin filtering the pixel. We calculate the original image’s row or column by multiplying the filtered image’s row or column by scale. For every **row** and **col** value pair there is a corresponding index in the filtered image where the first of the four bytes for the current pixel resides. This is calculated by multiplying the rowby the number of bytes per row and adding the column. This byte will hold the Red value, the next byte will hold the Blue value, and so on with the Green value and the Alpha value.For each pixel in the filtered image, the algorithm takes a **scale** by **scale** box of pixels from the original image and averages each component (Red, Green, Blue, and Alpha) and sets the results to each corresponding component of the new pixel.

The resulting filtered image will now be able to be displayed on any mobile device and still be readable.