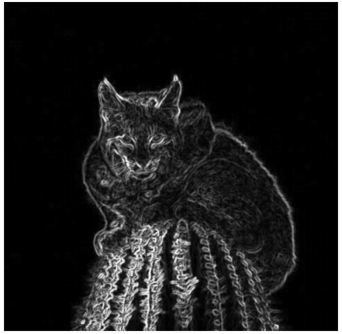


Image Processing

HTML5 – Canvas

Using Simple Image Filters

Brent M. Dingle, Ph.D.
Game Design and Development Program
Mathematics, Statistics and Computer Science
University of Wisconsin - Stout

2015

Lecture Objectives

- Provide Examples
  - Basic HTML5 canvas
  - Pixel Manipulation using Image Filters

- Bonus
  - Loading an image via drag-and-drop

What this is Not

- To complete your projects
  - You must learn more about HTML5 and JavaScript than what is about to be shown
    - This is an "on-your-own" activity
      - Instructor can help, but you must try on your own
    - A prereq to this course is CS 244
      - So you have programmed before
      - This stuff is "easy" compared to that =)
      - Likewise on the math topics
- In Sum: The following is just a place to start
  - More examples will follow throughout the course

Previously

- Digital Image Processing (DIP)
    - Is computer manipulation of pictures, or images, that have been converted into numeric form

- A Digital Image
    - Is a picture or image converted to numeric form
    - In grey-scale the image can be thought of as
        - 2D function f(x, y) or a matrix
        - x, y, and f(x, y) are discrete and finite
        - Image size = (x_{max}) by (y_{max}), e.g. 1024 x 768
        - Pixel Intensity Value = $f(x,y) \in [0, 255]$

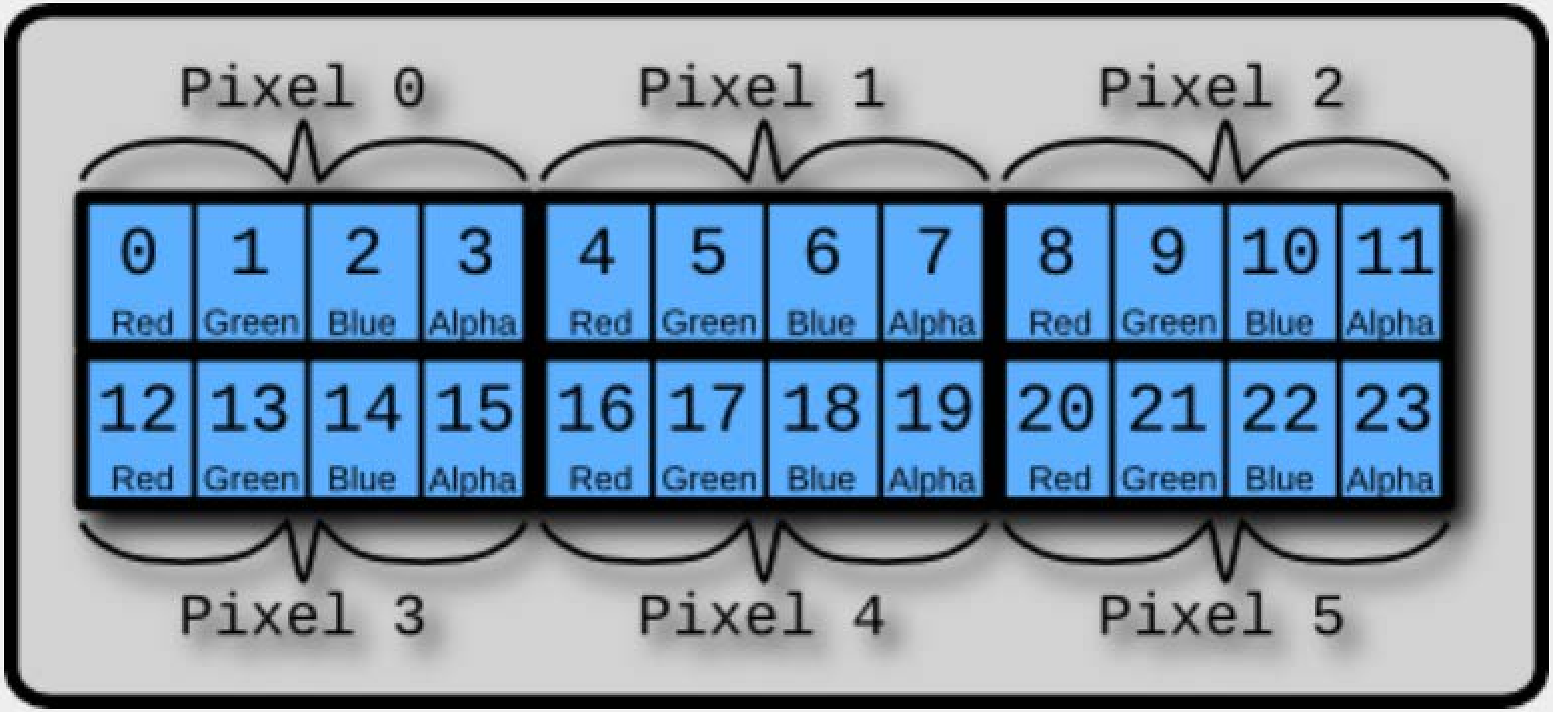
- Code to manipulate pixels directly already presented
    - Using HTML5 and JavaScript

Recall: Pixel Array

- JavaScript arrays work like C/C++/Java
  - Use the standard accessors to index into the array
    - EX:
      - mya[0] is the first element in the array named *mya*
      - mya[k-1] is the k-th element in the array named *mya*
  - Pixels in the array are in row-major order
    - with values of 0 to 255
    - where each four-integer group represents the four color channels: Red-Green-Blue-Alpha or RGBA

- *illustration next slide*

Pixel Order



Pixel layout in the pixel array for a 3-by-2 image of 6 pixels. Each pixel takes 4 elements in the array for red, green, blue, and alpha, for a total of 24 array elements, 0-23.

image from: http://beej.us/blog/data/html5s-canvas-2-pixel/

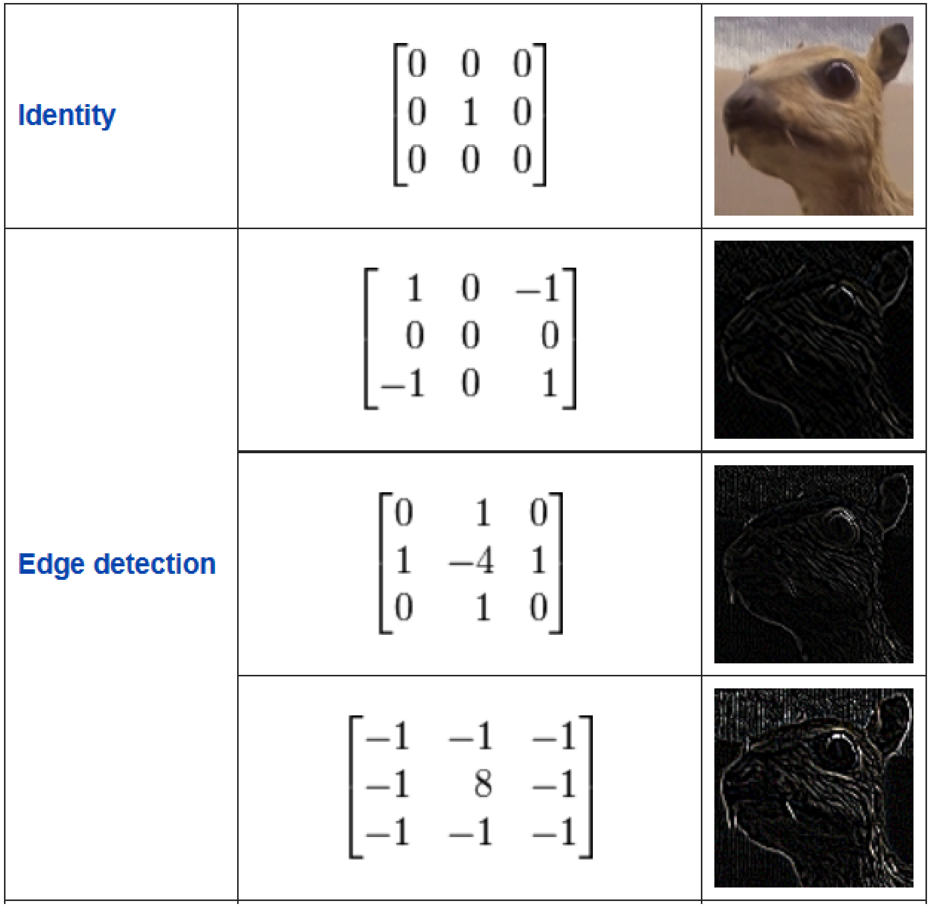
Today: Jump Ahead

- Image Filtering (and Edge detection) will be covered later
  - In the theoretical
  - It is used here as an example of using an Image Filter
  - Example code for doing this is more useful sooner than later

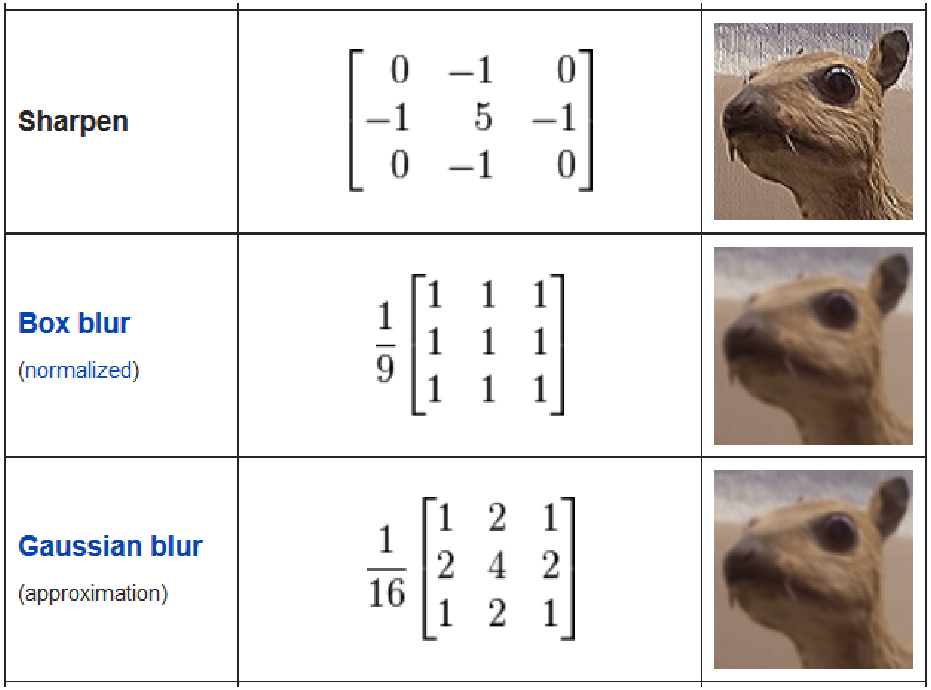
Kernels

- A ***kernel***, convolution matrix, or mask
  - is a 'small' matrix useful for image manipulation
    - edge-detection, blurring, sharpening, embossing…
  - This is accomplished by means of ***convolution*** between a kernel and an image
  - The size of the matrix reflects the size of the neighborhood by which a given pixel will be transformed

Examples



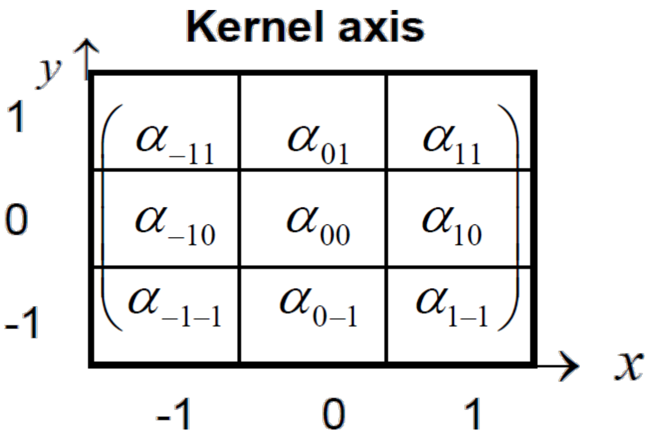
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	



Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

public domain images from Wikipedia

3x3 Convolution Math



$$\begin{pmatrix} \alpha_{-11} & \alpha_{01} & \alpha_{11} \\ \alpha_{-10} & \alpha_{00} & \alpha_{10} \\ \alpha_{-1-1} & \alpha_{0-1} & \alpha_{1-1} \end{pmatrix}$$

- 3 by 3 kernel with weights α_{ij}

The convolution of an *I_old* image by the kernel is denoted:

$$I\_new(x,y) = \sum_{j=-1}^{1} \sum_{i=-1}^{1} \alpha_{ij} I\_old(x-i, y-j)$$

$$I\_new\_normalized(x,y) = \frac{1}{\sum_{j=-1}^{1} \sum_{i=-1}^{1} \alpha_{ij}} \sum_{j=-1}^{1} \sum_{i=-1}^{1} \alpha_{ij} I\_old(x-i, y-j)$$

Aside: Normalization helps keep image values in the desired range
-- caution must be used in how and when this is done in program code to avoid undesired data loss, truncation, rounding, scaling…

Convolution: Pseudo-Code

- Pseudocode for the convolution of an ***image f(x,y)***
  with a ***kernel k(x, y)***
  to produce a ***new image g(x, y)***

```
for y = 0 to imageHeight
    for x = 0 to imageWidth
        sum = 0
        for i = -h to h
            for j = -w to w
                sum = sum + k(j, i) * f(x – j, y – i)
            end for j
        end for i
         g(x, y) = sum
    end for x
end for y
```

Assume:
kernel width is 2w+1
kernel height is 2h+1

Aside:
Be careful about non-symmetric kernels
Align their (-w, -h) to be same corner
as the image (0, 0)
i.e. upper left corner or lower left ?

Convolution Border Issues

- Some pixels do not have enough neighbors
  - typically: corner pixels and edge pixels

- Addressing these issues
  - Extend the image limits with zeros
  - Extend image limits with replication (of problem pixel value)
  - Generate specific filter for border cases

General Questions?

- Code example next

filterImage.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta content="text/html; charset=utf-8" http-equiv="content-type">
   <title>Example: Filter Loaded Image Data</title>
   <script src="filter.js" type="text/javascript"></script>
   <script src="filterImage.js" type="text/javascript"></script>
</head>

<body>
<div>
 <table>
      <tr>
          <canvas id="sourceCanvas" style="width:180px; height:90px; border:1px solid #aaaaaa; display:none">
             Your browser does not sup
          </canvas>
       </tr>
       <tr>
          <canvas id="outputCanvas" style="width:180px; height:90px; border:1px solid #aaaaaa;display:none">
          </canvas>
       </tr>
   </table></div>
</body>
</html>
```

Using TWO script files

And TWO canvas elements

- one to show input
- the other for output

Questions

- Questions on
  - filterImage HTML file ?
  - Next is filter image javaScript file

filterImage.js

```
var theProgram =
{
   // ------------------------------------------------------------------------
   // Pseudo-constants
   // ------------------------------------------------------------------------
   SOURCE_IMG_CANVAS_ID: "sourceCanvas",     // canvas id used in html
   OUTPUT_IMG_CANVAS_ID: "outputCanvas",     // canvas id used in html

   // ------------------------------------------------------------------------
   // Variables
   // ------------------------------------------------------------------------
   width:        400,         // canvas width... likely will be reset
   height:       400,         // canvas height... likely will be reset
   xOffset:       5,
   yOffset:       5,

   dropArea:      null,
   srcData:       null,

   destCanvas:    null,
   destCTX:       null,

   // ------------------------------------------------------------------------
   // Functions
```

“global” variables for the program

filterImage.js

```
Main: function()
 {
    // Setup the listeners
    this.dropArea = document.getElementById('theDropArea');
    this.dropArea.addEventListener('dragover', this.onDragOver, false);
    this.dropArea.addEventListener('drop', this.onDropFileSelect, false);

    // Get handle on the OUTPUT destCanvas
    theProgram.destCanvas
            = document.getElementById(theProgram.OUTPUT_IMG_CANVAS_ID)

    theProgram.destCTX = theProgram.destCanvas.getContext('2d');

    // initialize the filter
    theFilter.Init();

 },
```

Event handlers
to allow user to drag an input image file onto the canvas

filterImage.js

```
Main: function()
  {
     // Setup the listeners
     this.dropArea = document.getElementById('theDropArea');
     this.dropArea.addEventListener('dragover', this.onDragOver, false);
     this.dropArea.addEventListener('drop', this.onDropFileSelect, false);

     // Get handle on the OUTPUT destCanvas
     theProgram.destCanvas
             = document.getElementById(theProgram.OUTPUT_IMG_CANVAS_ID);

     theProgram.destCTX = theProgram.destCanvas.getContext('2d');

     // initialize the filter
     theFilter.Init();

  },

  // ---------------------------------------------------------------------
  // ---------------------------------------------------------------------
  onDragOver: function(evt)
  {
     evt.stopPropagation();
     evt.preventDefault();
     evt.dataTransfer.dropEffect = 'copy'; // Explicitly make this is a copy
  },
```

onDragOver

Give control of response solely to our program

Make drop event a ‘copy’ data

filterImage.js

```
  onDragOver: function(evt)
  {
    evt.stopPropagation();
    evt.preventDefault();
    evt.dataTransfer.dropEffect = 'copy'; // Explicitly make this is a copy
  },
// ----------------------------------------------------------------------------
  onDropFileSelect: function (evt)
  {
    evt.stopPropagation();
    evt.preventDefault();
    var files = evt.dataTransfer.files; // files that were dragged

    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
      var img = new Image;
      img.onload = function()
      {
        //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
        var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
        var ctx = canvas.getContext('2d');
        canvas.style.display = "block";
        canvas.width = img.width;
        canvas.height = img.height;
        canvas.style.width = canvas.width + "px" ;
        canvas.style.height = canvas.height + "px";
        ctx.drawImage(img, 0, 0);
```

onDrop

Again give full control to our program

filterImage.js

```
  onDragOver: function(evt)
  {
    evt.stopPropagation();
    evt.preventDefault();
    evt.dataTransfer.dropEffect = 'copy'; // Explicitly make this is a copy
  },
// ----------------------------------------------------------------------
  onDropFileSelect: function (evt)
  {
    evt.stopPropagation();
    evt.preventDefault();
    var files = evt.dataTransfer.files; // files that were dragged

    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
      var img = new Image;
      img.onload = function()
      {
        //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
        var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
        var ctx = canvas.getContext('2d');
        canvas.style.display = "block";
        canvas.width = img.width;
        canvas.height = img.height;
        canvas.style.width = canvas.width + "px" ;
        canvas.style.height = canvas.height + "px";
        ctx.drawImage(img, 0, 0);
```

Get the list of files
that were dropped on the canvas

filterImage.js

```
  onDragOver: function(evt)
  {
    evt.stopPropagation();
    evt.preventDefault();
    evt.dataTransfer.dropEffect = 'copy'; // Explicitly make this is a copy
  },
// ---------------------------------------------------------------------
  onDropFileSelect: function (evt)
  {
    evt.stopPropagation();
    evt.preventDefault();
    var files = evt.dataTransfer.files; // files that were dragged

    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
      var img = new Image;
      img.onload = function()
      {
        //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
        var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
        var ctx = canvas.getContext('2d');
        canvas.style.display = "block";
        canvas.width = img.width;
        canvas.height = img.height;
        canvas.style.width = canvas.width + "px" ;
        canvas.style.height = canvas.height + "px";
        ctx.drawImage(img, 0, 0);
```

Only look at first file in list → item index ZERO

If that file type is not an image file
Then we do nothing (drop results in nothing)

filterImage.js

```
  onDragOver: function(evt)
  {
    evt.stopPropagation();
    evt.preventDefault();
    evt.dataTransfer.dropEffect = 'copy'; // Explicitly make this is a copy
  },
// ----------------------------------------------------------------------
  onDropFileSelect: function (evt)
  {
    evt.stopPropagation();
    evt.preventDefault();
    var files = evt.dataTransfer.files; // files that were dragged

    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
      var img = new Image;
      img.onload = function()
      {
        //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
        var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
        var ctx = canvas.getContext('2d');
        canvas.style.display = "block";
        canvas.width = img.width;
        canvas.height = img.height;
        canvas.style.width = canvas.width + "px" ;
        canvas.style.height = canvas.height + "px";
        ctx.drawImage(img, 0, 0);
```

Allocate memory for a new image

filterImage.js

```
   onDragOver: function(evt)
   {
      evt.stopPropagation();
      evt.preventDefault();
      evt.dataTransfer.dropEffect = 'copy'; // Explicitly make this is a copy
   },
// ----------------------------------------------------------------------
   onDropFileSelect: function (evt)
   {
      evt.stopPropagation();
      evt.preventDefault();
      var files = evt.dataTransfer.files; // files that were dragged

      // If the "first" file is not an image, do nothing
      var curFile = files[0];
      // Only process image file
      if ( curFile.type.match('image.*') )
      {
         var img = new Image();
         img.onload = function()
         {
            //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
            var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
            var ctx = canvas.getContext('2d');
            canvas.style.display = "block";
            canvas.width = img.width;
            canvas.height = img.height;
            canvas.style.width = canvas.width + "px" ;
            canvas.style.height = canvas.height + "px";
            ctx.drawImage(img, 0, 0);
```

Set up a load function for the new image
BEFORE we assign a file source name
to the image

filterImage.js

```
    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
        var img = new Image;
        img.onload = function()
        {
            //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
            var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
            var ctx = canvas.getContext('2d');
            canvas.style.display = "block";
            canvas.width = img.width;
            canvas.height = img.height;
            canvas.style.width = canvas.width + "px" ;
            canvas.style.height = canvas.height + "px";
            ctx.drawImage(img, 0, 0);

            // Set the source data BEFORE applying any filters!
            theProgram.srcData  = ctx.getImageData(0, 0, img.width, img.height);

            theProgram.applyConvSobel();

            URL.revokeObjectURL(img.src);  // clear up memory...
        }
       img.src = URL.createObjectURL(curFile);
    }
    // else current file type is NOT image --> so do nothing
},
```

Get canvas and context for the SOURCE image

filterImage.js

```
    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
        var img = new Image;
        img.onload = function()
        {
            //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
            var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
            var ctx = canvas.getContext('2d');
            canvas.style.display = "block";
            canvas.width = img.width;
            canvas.height = img.height;
            canvas.style.width = canvas.width + "px" ;
            canvas.style.height = canvas.height + "px";
            ctx.drawImage(img, 0, 0);

            // Set the source data BEFORE applying any filters!
            theProgram.srcData  = ctx.getImageData(0, 0, img.width, img.height);

            theProgram.applyConvSobel();

            URL.revokeObjectURL(img.src);  // clear up memory...
        }
       img.src = URL.createObjectURL(curFile);
    }
    // else current file type is NOT image --> so do nothing
},
```

Set display style to block (make canvas visible)
Size everything correctly

filterImage.js

```
  // If the "first" file is not an image, do nothing
  var curFile = files[0];
  // Only process image file
  if ( curFile.type.match('image.*') )
  {
    var img = new Image;
    img.onload = function()
    {
      //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
      var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
      var ctx = canvas.getContext('2d');
      canvas.style.display = "block";
      canvas.width = img.width;
      canvas.height = img.height;
      canvas.style.width = canvas.width + "px" ;
      canvas.style.height = canvas.height + "px";
      ctx.drawImage(img, 0, 0);

      // Set the source data BEFORE applying any filters!
      theProgram.srcData  = ctx.getImageData(0, 0, img.width, img.height);

      theProgram.applyConvSobel();

      URL.revokeObjectURL(img.src);  // clear up memory...
    }
    img.src = URL.createObjectURL(curFile);
  }
  // else current file type is NOT image --> so do nothing
},
```

Draw image
starting at canvas coordinates (0, 0)

filterImage.js

```
    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
       var img = new Image;
       img.onload = function()
       {
          //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
          var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
          var ctx = canvas.getContext('2d');
          canvas.style.display = "block";
          canvas.width = img.width;
          canvas.height = img.height;
          canvas.style.width = canvas.width + "px" ;
          canvas.style.height = canvas.height + "px";
          ctx.drawImage(img, 0, 0);

          // Set the source data BEFORE applying any filters!
          theProgram.srcData  = ctx.getImageData(0, 0, img.width, img.height);

          theProgram.applyConvSobel();

          URL.revokeObjectURL(img.src);  // clear up memory...
       }
      img.src = URL.createObjectURL(curFile);
    }
    // else current file type is NOT image --> so do nothing
 },
```

Set the global variable to reference the source image data (as drawn on canvas)

filterImage.js

```
    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
      var img = new Image;
      img.onload = function()
      {
        //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
        var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
        var ctx = canvas.getContext('2d');
        canvas.style.display = "block";
        canvas.width = img.width;
        canvas.height = img.height;
        canvas.style.width = canvas.width + "px" ;
        canvas.style.height = canvas.height + "px";
        ctx.drawImage(img, 0, 0);

        // Set the source data BEFORE applying any filters!
        theProgram.srcData  = ctx.getImageData(0, 0, img.width, img.height);

        theProgram.applyConvSobel();

        URL.revokeObjectURL(img.src);  // clear up memory...
      }
      img.src = URL.createObjectURL(curFile);
    }
    // else current file type is NOT image --> so do nothing
  },
```

Apply the desired filter

filterImage.js

```
    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
        var img = new Image;
        img.onload = function()
        {
            //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
            var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
            var ctx = canvas.getContext('2d');
            canvas.style.display = "block";
            canvas.width = img.width;
            canvas.height = img.height;
            canvas.style.width = canvas.width + "px" ;
            canvas.style.height = canvas.height + "px";
            ctx.drawImage(img, 0, 0);

            // Set the source data BEFORE applying any filters!
            theProgram.srcData  = ctx.getImageData(0, 0, img.width, img.height);

            theProgram.applyConvSobel();

            URL.revokeObjectURL(img.src);  // clear up memory...
        }
        img.src = URL.createObjectURL(curFile);
    }
    // else current file type is NOT image --> so do nothing
},
```

Free up some memory

filterImage.js

```
    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
        var img = new Image;
        img.onload = function()
        {
            //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
            var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
            var ctx = canvas.getContext('2d');
            canvas.style.display = "block";
            canvas.width = img.width;
            canvas.height = img.height;
            canvas.style.width = canvas.width + "px" ;
            canvas.style.height = canvas.height + "px";
            ctx.drawImage(img, 0, 0);

            // Set the source data BEFORE applying any filters!
            theProgram.srcData  = ctx.getImageData(0, 0, img.width, img.height);

            theProgram.applyConvSobel();

            URL.revokeObjectURL(img.src);  // clear up memory
        }
       img.src = URL.createObjectURL(curFile);
    }
    // else current file type is NOT image --> so do nothing
},
```

the img.onload() function ends

filterImage.js

```
    // If the "first" file is not an image, do nothing
    var curFile = files[0];
    // Only process image file
    if ( curFile.type.match('image.*') )
    {
       var img = new Image;
       img.onload = function()
       {
          //theProgram.dropArea.style.display = "none"; unrem to allow only 1 drop
          var canvas = document.getElementById(theProgram.SOURCE_IMG_CANVAS_ID);
          var ctx = canvas.getContext('2d');
          canvas.style.display = "block";
          canvas.width = img.width;
          canvas.height = img.height;
          canvas.style.width = canvas.width + "px" ;
          canvas.style.height = canvas.height + "px";
          ctx.drawImage(img, 0, 0);

          // Set the source data BEFORE applying any filters!
          theProgram.srcData  = ctx.getImageData(0, 0, img.widt

          theProgram.applyConvSobel();

          URL.revokeObjectURL(img.src);  // clear up memory...
       }
      img.src = URL.createObjectURL(curFile);
    }
    // else current file type is NOT image --> so do nothing
 },
```

NOW Set the source file of our image

- this will trigger the onload function to be called when the file is completely loaded into the browser
- thus the onload function must be defined BEFORE doing this

filterImage.js

```
// ----------------------------------------------------------------------
applyConvIdentity: function()
{
   // below should do 'nothing' apply a filter but change nothing -- debug test
   var destData = theFilter.convolute(theProgram.srcData,
                              [  0,  0,  0,
                                 0,  1,  0,
                                 0,  0,  0   ]);
   theProgram.displayOutput(destData, theProgram.srcData);
},
```

applyConvIdentity
   Apply an identity filter to test the filter class
   This filter/matrix should change nothing
    The end effect is
            pixel[x, y] = pixel[x, y] + 0;

All α_{ij} = 0 except α_{00} = 1

$$I\_new(x, y) = \sum_{j=-1}^{1} \sum_{i=-1}^{1} \alpha_{ij} I\_old(x-i, y-j)$$

filterImage.js

```
// Sobel based edge detection
// Notice this works in steps: greyscale, horizontal, vertical, combine
// ---------------------------------------------------------------------
  applyConvSobel: function()
  {
    var grey = theFilter.greyscale(theProgram.srcData);

    var vertical = theFilter.convoluteFloat32(grey,
                                     [-1, -2, -1,
                                       0,  0,  0,
                                       1,  2,  1]  );
    var horizontal = theFilter.convoluteFloat32(grey,
                                     [-1,  0,  1,
                                      -2,  0,  2,
                                      -1,  0,  1]  );

    var finalImg = theProgram.destCTX.createImageData(vertical.width, vertical.height);

    for (var i=0; i<finalImg.data.length; i+=4)
    {
      var v = Math.abs(vertical.data[i]);
      var h = Math.abs(horizontal.data[i]);
      finalImg.data[i] = (v+h)/2;
      finalImg.data[i+1] = (v+h)/2;
      finalImg.data[i+2] = (v+h)/2;
      finalImg.data[i+3] = 255;
    }

    theProgram.displayOutput(finalImg, theProgram.srcData);
  },
```

Sobel filter implemented as 4 steps

filterImage.js

```
// Sobel based edge detection
// Notice this works in steps: greyscale, horizontal, vertical, combine
// -----------------------------------------------------------------------
   applyConvSobel: function()
   {
      var grey = theFilter.greyscale(theProgram.srcData);

      var vertical = theFilter.convoluteFloat32(grey,
                                          [-1, -2, -1,
                                            0,  0,  0,
                                            1,  2,  1]  );
      var horizontal = theFilter.convoluteFloat32(grey,
                                          [-1,  0,  1,
                                           -2,  0,  2,
                                           -1,  0,  1]  );

      var finalImg = theProgram.destCTX.createImageData(vertical.width, vertical.height);

      for (var i=0; i<finalImg.data.length; i+=4)
      {
         var v = Math.abs(vertical.data[i]);
         var h = Math.abs(horizontal.data[i]);
         finalImg.data[i] = (v+h)/2;
         finalImg.data[i+1] = (v+h)/2;
         finalImg.data[i+2] = (v+h)/2;
         finalImg.data[i+3] = 255;
      }

      theProgram.displayOutput(finalImg, theProgram.srcData);
   },
```

Step 1

Greyscale the entire image

This is using a *greyscale()* function defined in filter.js

We will look at that a little later

filterImage.js

```
// Sobel based edge detection
// Notice this works in steps: greyscale, horizontal, vertical, combine
// ---------------------------------------------------------------------
   applyConvSobel: function()
   {
      var grey = theFilter.greyscale(theProgram.srcData);

      var vertical = theFilter.convoluteFloat32(grey,
                                        [-1, -2, -1,
                                          0,  0,  0,
                                          1,  2,  1]  );
      var horizontal = theFilter.convoluteFloat32(grey,
                                        [-1,  0,  1,
                                         -2,  0,  2,
                                         -1,  0,  1]  );

      var finalImg = theProgram.destCTX.createImageData(vertical.width, vertical.height);

      for (var i=0; i<finalImg.data.length; i+=4)
      {
         var v = Math.abs(vertical.data[i]);
         var h = Math.abs(horizontal.data[i]);
         finalImg.data[i] = (v+h)/2;
         finalImg.data[i+1] = (v+h)/2;
         finalImg.data[i+2] = (v+h)/2;
         finalImg.data[i+3] = 255;
      }

      theProgram.displayOutput(finalImg, theProgram.srcData);
   },
```

Step 2

Perform a VERTICAL filtering
of the greyscaled image
Store result in variable named *vertical*

filterImage.js

```
// Sobel based edge detection
// Notice this works in steps: greyscale, horizontal, vertical, combine
// ----------------------------------------------------------------------
  applyConvSobel: function()
  {
    var grey = theFilter.greyscale(theProgram.srcData);

    var vertical = theFilter.convoluteFloat32(grey,
                                   [-1, -2, -1,
                                     0,  0,  0,
                                     1,  2,  1]  );
    var horizontal = theFilter.convoluteFloat32(grey,
                                   [-1,  0,  1,
                                    -2,  0,  2,
                                    -1,  0,  1]  );

    var finalImg = theProgram.destCTX.createImageData(vertical.width, vertical.height);

    for (var i=0; i<finalImg.data.length; i+=4)
    {
      var v = Math.abs(vertical.data[i]);
      var h = Math.abs(horizontal.data[i]);
      finalImg.data[i] = (v+h)/2;
      finalImg.data[i+1] = (v+h)/2;
      finalImg.data[i+2] = (v+h)/2;
      finalImg.data[i+3] = 255;
    }

    theProgram.displayOutput(finalImg, theProgram.srcData);
  },
```

Step 3

Perform a HORIZONTAL filtering
of the greyscaled image
Store result in variable named *horizontal*

filterImage.js

```
// Sobel based edge detection
// Notice this works in steps: greyscale, horizontal, vertical, combine
// ----------------------------------------------------------------------
   applyConvSobel: function()
   {
      var grey = theFilter.greyscale(theProgram.srcData);

      var vertical = theFilter.convoluteFloat32(grey,
                                   [-1, -2, -1,
                                     0,  0,  0,
                                     1,  2,  1]  );
      var horizontal = theFilter.convoluteFloat32(grey,
                                   [-1,  0,  1,
                                    -2,  0,  2,
                                    -1,  0,  1]  );

      var finalImg = theProgram.destCTX.createImageData(vertical.width, vertical.height);

      for (var i=0; i<finalImg.data.length; i+=4)
      {
         var v = Math.abs(vertical.data[i]);
         var h = Math.abs(horizontal.data[i]);
         finalImg.data[i] = (v+h)/2;
         finalImg.data[i+1] = (v+h)/2;
         finalImg.data[i+2] = (v+h)/2;
         finalImg.data[i+3] = 255;
      }

      theProgram.displayOutput(finalImg, theProgram.srcData);
   },
```

Step 4a

Allocate memory for final image

filterImage.js

```
// Sobel based edge detection
// Notice this works in steps: greyscale, horizontal, vertical, combine
// ----------------------------------------------------------------------
   applyConvSobel: function()
   {
      var grey = theFilter.greyscale(theProgram.srcData);

      var vertical = theFilter.convoluteFloat32(grey,
                                    [-1, -2, -1,
                                      0,  0,  0,
                                      1,  2,  1]  );
      var horizontal = theFilter.convoluteFloat32(grey,
                                    [-1,  0,  1,
                                     -2,  0,  2,
                                     -1,  0,  1]  );

      var finalImg = theProgram.destCTX.createImageData(vertical.width, vertical.height);

      for (var i=0; i<finalImg.data.length; i+=4)
      {
         var v = Math.abs(vertical.data[i]);
         var h = Math.abs(horizontal.data[i]);
         finalImg.data[i] = (v+h)/2;
         finalImg.data[i+1] = (v+h)/2;
         finalImg.data[i+2] = (v+h)/2;
         finalImg.data[i+3] = 255;
      }

      theProgram.displayOutput(finalImg, theProgram.srcData);
   },
```

Step 4b

Set the data values of final image

- Red = average of horizontal and vertical filter
- Green = average of horizontal and vertical filter
- Blue = average of horizontal and vertical filter
- Alpha = 255 (opaque)

filterImage.js

```
// Sobel based edge detection
// Notice this works in steps: greyscale, horizontal, vertical, combine
// -------------------------------------------------------------------
   applyConvSobel: function()
   {
      var grey = theFilter.greyscale(theProgram.srcData);

      var vertical = theFilter.convoluteFloat32(grey,
                                   [-1, -2, -1,
                                     0,  0,  0,
                                     1,  2,  1]  );
      var horizontal = theFilter.convoluteFloat32(grey,
                                   [-1,  0,  1,
                                    -2,  0,  2,
                                    -1,  0,  1]  );

      var finalImg = theProgram.destCTX.createImageData(vertical.width, vertical.height);

      for (var i=0; i<finalImg.data.length; i+=4)
      {
         var v = Math.abs(vertical.data[i]);
         var h = Math.abs(horizontal.data[i]);
         finalImg.data[i] = (v+h)/2;
         finalImg.data[i+1] = (v+h)/2;
         finalImg.data[i+2] = (v+h)/2;
         finalImg.data[i+3] = 255;
      }

      theProgram.displayOutput(finalImg, theProgram.srcData);
   },
```

Call a function to display the result
to the correct canvas and context

filterImage.js

```
// ----------------------------------------------------------------------
   displayOutput: function(destData)
   {
      // make things visible and correct size
      theProgram.destCanvas.style.display = "block";
      theProgram.destCanvas.width = destData.width;
      theProgram.destCanvas.height = destData.height;
      theProgram.destCanvas.style.width = theProgram.destCanv
      theProgram.destCanvas.style.height = theProgram.destCanv

      theProgram.destCTX.putImageData(destData, 0, 0);
   },


};  // end theProgram variable
```

Make the destination canvas visible with block display

filterImage.js

```
// -----------------------------------------------------------------------
   displayOutput: function(destData)
   {
      // make things visible and correct size
      theProgram.destCanvas.style.display = "block";
      theProgram.destCanvas.width = destData.width;
      theProgram.destCanvas.height = destData.height;
      theProgram.destCanvas.style.width = theProgram.destCanvas.width + "px" ;
      theProgram.destCanvas.style.height = theProgram.destCanvas.height + "px";

      theProgram.destCTX.putImageData(destData, 0, 0);
   },


};  // end theProgram variable
```

Size the canvas to image size

filterImage.js

```
// ----------------------------------------------------------------------
   displayOutput: function(destData)
   {
      // make things visible and correct size
      theProgram.destCanvas.style.display = "block";
      theProgram.destCanvas.width = destData.width;
      theProgram.destCanvas.height = destData.height;
      theProgram.destCanvas.style.width = theProgram.destCanvas.width + "px" ;
      theProgram.destCanvas.style.height = theProgram.destCanvas.height + "px";

      theProgram.destCTX.putImageData(destData, 0, 0);
   },


};  // end theProgram variable

// -------------------------------------------------------------------------
window.onload = function()
{
   // Initialize and Start the game
   theProgram.Main();

};
```

And run the program
once everything on the page is loaded

Questions

- Questions on
  - filterImage.js javaScript file ?
  - next is filter.js javaScript file

filter.js

```
var theFilter =
{
   // -----------------------------------------------------------------------
   // Pseudo-constants
   // -----------------------------------------------------------------------

   // -----------------------------------------------------------------------
   // Variables
   // -----------------------------------------------------------------------

   // -----------------------------------------------------------------------
   // Functions
   // -----------------------------------------------------------------------
   //                                          Init
   // -----------------------------------------------------------------------
   Init: function()
   {
      // nothing needs to be done here... yet =)
   },
```

Start easy

No class variables

and

No initialization needed

filter.js

```
var theFilter =
{
                              ::
                              ::
  // -----------------------------------------------------------------------
  // greyscale
  // Assumed:
  //     pixels are data struct returned by: ctx.getImageData(0, 0, w, h)
  //
  greyscale: function(pixels)
  {
    var d = pixels.data;
    for (var i=0; i < d.length; i+=4)
    {
      var r = d[i];
      var g = d[i+1];
      var b = d[i+2];
      // CIE luminance for the RGB --- fixes appearance for humans
      var v = 0.2126*r + 0.7152*g + 0.0722*b;
      d[i] = d[i+1] = d[i+2] = v
    }
    return pixels;
  },
```

Many filters will need to first produce a greyscale version of the image

So it is included in this class

Note there is an implication in applying a human eyesight based luminosity adjustment here

Some filters might not want that done…

Also note
variable ***d***
is acting as a name alias for pixels.data

So pixels.data is being altered by this function
HOWEVER
pixels itself is a copy of the data sent as a parameter to this function
i.e. pixels is a "primitive" JS type, (array-string-ish)
or rather pixels is not an "object"
so when returned here it is to modified data, not the original passed in values

filter.js

```
// This function is needed for doing things like a Sobel filter
// where intermediate steps do not work if values are clamped
// ------------------------------------------------------------------------
    convoluteFloat32: function(pixels, weights, opaque)
    {
        if (!window.Float32Array)
        {
            Float32Array = Array;
        }
        var side = Math.round(Math.sqrt(weights.length));
        var halfSide = Math.floor(side/2);

        var src = pixels.data;
        var sw = pixels.width;
        var sh = pixels.height;

        var w = sw;
        var h = sh;
        var output = {  width: w,
                        height: h,
                        data: new Float32Array(w*h*4)
                      };

        var dst = output.data;

        var alphaFac = opaque ? 1 : 0;
```

This was necessary to support
“older” browsers (for example: IE9)
It might not be needed now

filter.js

```
 // This function is needed for doing things like a Sobel filter
// where intermediate steps do not work if values are clamped
// ----------------------------------------------------------------------
    convoluteFloat32: function(pixels, weights, opaque)
    {
        if (!window.Float32Array)
        {
            Float32Array = Array;
        }
        var side = Math.round(Math.sqrt(weights.length));
        var halfSide = Math.floor(side/2);

        var src = pixels.data;
        var sw = pixels.width;
        var sh = pixels.height;

        var w = sw;
        var h = sh;
        var output = { width: w,
                        height: h,
                        data: new Float32Array(w*h*4)
                     };

        var dst = output.data;

        var alphaFac = opaque ? 1 : 0;
```

Setup size of things

weights.length
is the size of the kernel matrix
e.g. 3x3 matrix has length = 9

filter.js

```
// This function is needed for doing things like a Sobel filter
// where intermediate steps do not work if values are clamped
// ----------------------------------------------------------------------
   convoluteFloat32: function(pixels, weights, opaque)
   {
      if (!window.Float32Array)
      {
         Float32Array = Array;
      }
      var side = Math.round(Math.sqrt(weights.length));
      var halfSide = Math.floor(side/2);

      var src = pixels.data;
      var sw = pixels.width;
      var sh = pixels.height;

      var w = sw;
      var h = sh;
      var output = {  width: w,
                      height: h,
                      data: new Float32Array(w*h*4)
                   };

      var dst = output.data;

      var alphaFac = opaque ? 1 : 0;
```

Allocate memory for the resulting output
output.w = width of resulting image
output.h = height of resulting image
output.data = pixel data… as type float32 (not integer)

filter.js

```
 // This function is needed for doing things like a Sobel filter
// where intermediate steps do not work if values are clamped
// -----------------------------------------------------------------------
   convoluteFloat32: function(pixels, weights, opaque)
   {
      if (!window.Float32Array)
      {
         Float32Array = Array;
      }
      var side = Math.round(Math.sqrt(weights.length));
      var halfSide = Math.floor(side/2);

      var src = pixels.data;
      var sw = pixels.width;
      var sh = pixels.height;

      var w = sw;
      var h = sh;
      var output = {  width: w,
                      height: h,
                      data: new Float32Array(w*h*4)
                   };

      var dst = output.data;

      var alphaFac = opaque ? 1 : 0;
```

dst is name alias for output.data → saves some typing

filter.js

```
 // This function is needed for doing things like a Sobel filter
// where intermediate steps do not work if values are clamped
// ----------------------------------------------------------------------
   convoluteFloat32: function(pixels, weights, opaque)
   {
      if (!window.Float32Array)
      {
         Float32Array = Array;
      }
      var side = Math.round(Math.sqrt(weights.length));
      var halfSide = Math.floor(side/2);

      var src = pixels.data;
      var sw = pixels.width;
      var sh = pixels.height;

      var w = sw;
      var h = sh;
      var output = {  width: w,
                      height: h,
                      data: new Float32Array(w*h*4)
                   };

      var dst = output.data;

      var alphaFac = opaque ? 1 : 0;
```

Convert Boolean to integer
Also covers the case if opaque parameter was NOT sent
→ sets alphaFac to 1

filter.js

```
    var alphaFac = opaque ? 1 : 0;
    for (var y=0; y<h; y++)
    {
      for (var x=0; x<w; x++)
      {
        var sy = y;
        var sx = x;
        var dstOff = (y*w+x)*4;
        var r=0, g=0, b=0, a=0;
        for (var cy=0; cy<side; cy++)
        {
          for (var cx=0; cx<side; cx++)
          {
            var scy = Math.min(sh-1, Math.max(0, sy + cy - halfSide));
            var scx = Math.min(sw-1, Math.max(0, sx + cx - halfSide));
            var srcOff = (scy*sw+scx)*4;
            var wt = weights[cy*side+cx];
            r += src[srcOff] * wt;
            g += src[srcOff+1] * wt;
            b += src[srcOff+2] * wt;
            a += src[srcOff+3] * wt;
          }
        }
        dst[dstOff] = r;
        dst[dstOff+1] = g;
        dst[dstOff+2] = b;
        dst[dstOff+3] = a + alphaFac*(255-a);
      }
    }
    return output;
  },
};
```

Non-rounding/truncating code for the pseudo-code:

```
for y = 0 to imageHeight
   for x = 0 to imageWidth
      sum = 0
      for i = -h to h
         for j = -w to w
            sum = sum + k(j, i) * f(x – j, y – i)
         end for j
      end for i
       g(x, y) = sum
   end for x
end for y
```

Questions?

- Questions on the filter.js JavaScript file?

- Full example code available online
  - with extras

Other Questions?

- Beyond D2L
  - Examples and information can be found online at:
    - *http://docdingle.com/teaching/cs.html*
    - *Continue to more stuff as needed*

Extra Reference Stuff Follows

Credits

- Much of the content derived/based on slides for use with the book:
  - *Digital Image Processing,* Gonzalez and Woods

- Some layout and presentation style derived/based on presentations by
  - Donald House, Texas A&M University, 1999
  - Bernd Girod, Stanford University, 2007
  - Shreekanth Mandayam, Rowan University, 2009
  - Igor Aizenberg, TAMUT, 2013
  - Xin Li, WVU, 2014
  - George Wolberg, City College of New York, 2015
  - Yao Wang and Zhu Liu, NYU-Poly, 2015
  - Sinisa Todorovic, Oregon State, 2015
  - Beej's Bit Bucket / Tech and Programming Fun
    - http://beej.us/blog/
    - http://beej.us/blog/data/html5s-canvas-2-pixel/
  - w3schools.com