MCMASTER UNIVERSITY

3SK3 COURSE PROJECT 1

IMAGE INTERPOLATION

KHALID ASAD
1147299
MARCH 3RD 2014

Algorithm:

1. Read each pixel value by reading every column and row.
2. Find approximation of partial derivatives for each pixel using slope equations from lecture slides.
   $$\partial_x f(x,y) = [f(x+1,y) - f(x-1,y)]/2$$
   $$\partial_y f(x,y) = [f(x,y+1) - f(x,y-1)]/2$$
   $$\partial_{xy} f(x,y) = [f(x+1,y+1) - f(x-1,y) - f(x,y-1) + f(x,y)]/4$$
3. Create a new matrix based on the old one multiplied by a factor.
4. Find width and heights for new matrix in the current row and column.
5. Find 16 coefficients by reading specific points in the partial derivatives and original matrix
6. By grouping all of the coefficients into one matrix, alpha, we can use an equation to find the formula for alpha. $\alpha = \begin{bmatrix} a00 & a10 & a20 & a30 & a01 & a11 & a21 & a31 & a02 & a12 & a22 & a32 & a03 & a13 & a23 & a33 \end{bmatrix}^T$
7. Rearranging the equation and with a given A inverse, we can finally solve for alpha. $A\alpha = x$

$$A^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 \\ -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 \\ 9 & -9 & -9 & 9 & 6 & 3 & -6 & -3 & 6 & -6 & 3 & -3 & 4 & 2 & 2 & 1 \\ -6 & 6 & 6 & -6 & -3 & -3 & 3 & 3 & -4 & 4 & -2 & 2 & -2 & -2 & -1 & -1 \\ 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ -6 & 6 & 6 & -6 & -4 & -2 & 4 & 2 & -3 & 3 & -3 & 3 & -2 & -1 & -2 & -1 \\ 4 & -4 & -4 & 4 & 2 & 2 & -2 & -2 & 2 & -2 & 2 & -2 & 1 & 1 & 1 & 1 \end{bmatrix}$$

8. Now with the value of alpha, we can calculate temporary width and height values of a 4x4 pixel.
9. Place the pixel sum value in the new image matrix shifted downright by 1.
10. Repeat steps 4-9 for every row and column. This keeps updated alpha values for a more precise image interpolation.

Operations:

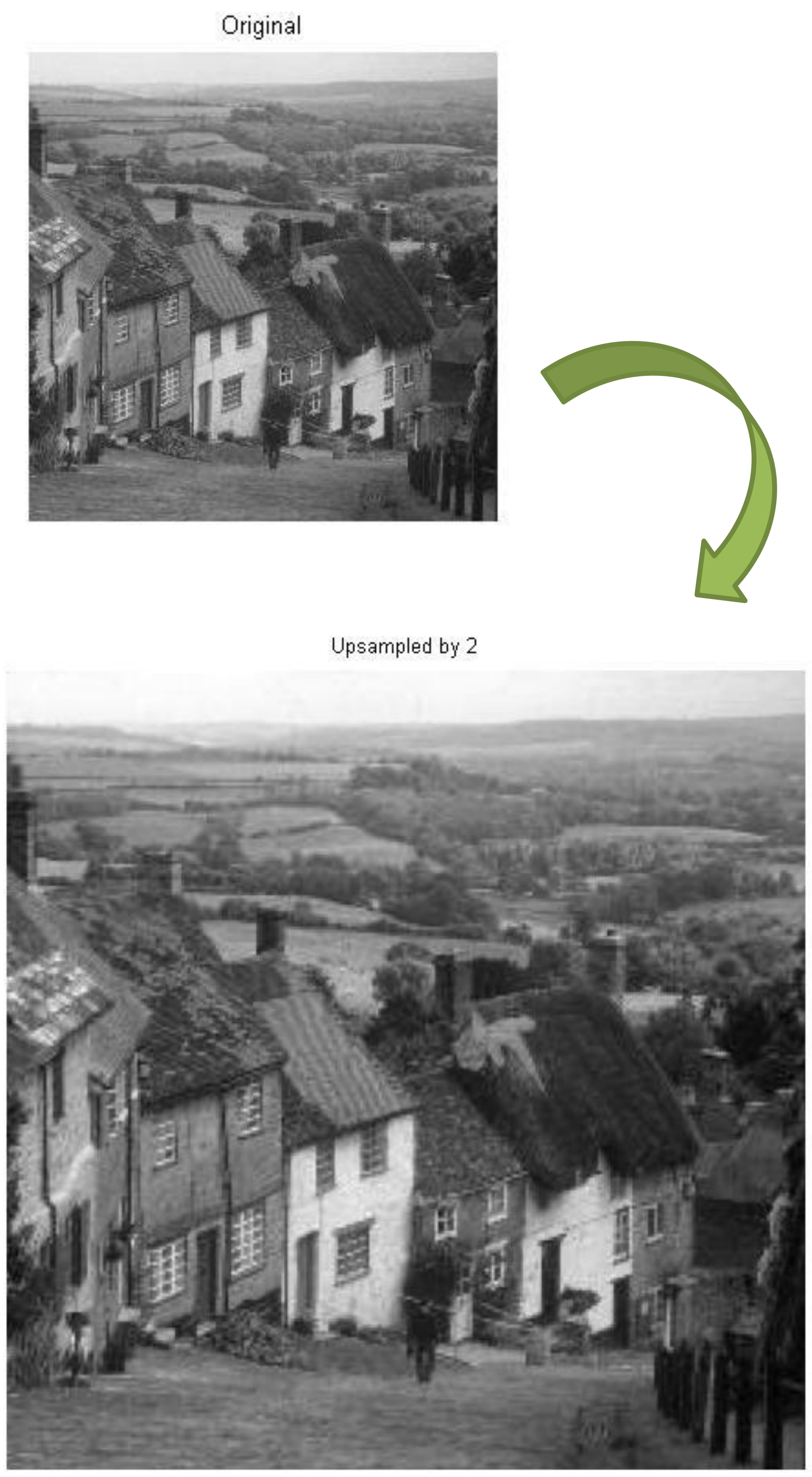
16 additions, 2*16 divisions, 2*16 multiplications = **80** operations
The amount of operations needed to find the center missing pixels is the same as interpolating other pixels.

Complexity:

The algorithm complexity is O(n^2) due to the fact that there are 2 instances of nested loops. O(n^2) + 16*O(n^2) = 17* O(n^2).
Taking a 256x256 image and interpolating by a factor of 2 takes 4.143 seconds. Interpolating the same image twice by a factor of 2 (interpolating a 512x512 image) takes 20.843 seconds.
20.843 - 4.143 = 16.7 16.7/4.143 = 4.031 4.031 ~= 4.143 16.7 ~= 4^2
256*256 = 65536 512*512 = 262144 262144/65536 = 4 ~=4.031~=4.143
Therefore the algorithm is indeed O(n^2).

Upsample:

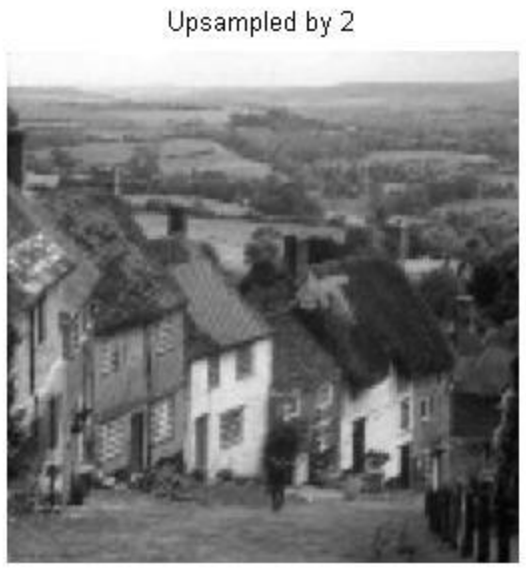
We interpolate our original image shown below to twice the size. This needed to be converted to grayscale using the rgb2gray function which is commented out in the MATLAB program. This image is 'city.jpg'



The results are quite astonishing. With only 4.386 seconds of time, this 256x256 image was interpolated to a 512x512 image.

Downsample:

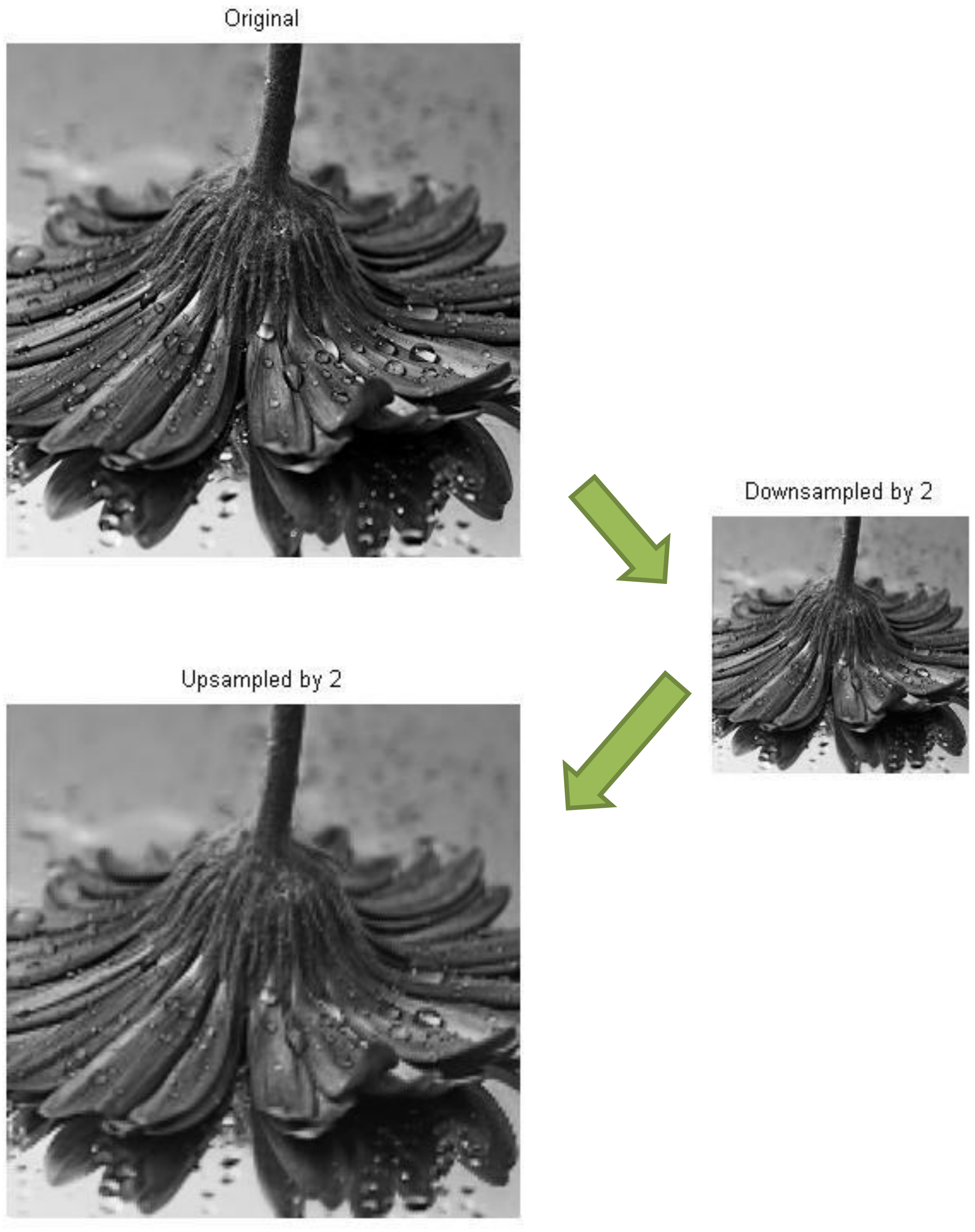
Now we will downsample the same image by a factor of 2, upsample it and then compare it with the original.



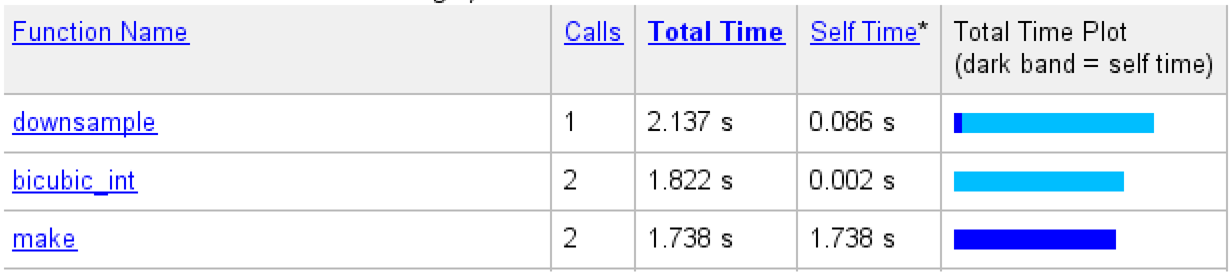
The image was downsampled by dropping every other row and column and was upsampled to original size by the interpolation algorithm. The new picture is significantly grainy because of loss of data. The Mean Squared Error turns out to be 20.3940. Here are the timing results:

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
downsample	1	1.646 s	0.086 s	▬
bicubic_int	2	1.321 s	0.002 s	▬
make	2	1.257 s	1.257 s	▬

Again we will try downsampling and upsampling with another image (300x300):



Again the loss of data is clearly evident. The Mean Squared Error is 30.9277. Here are the timing results:



Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
downsample	1	2.137 s	0.086 s	▬
bicubic_int	2	1.822 s	0.002 s	▬
make	2	1.738 s	1.738 s	▬

Conclusion:

The algorithm produced performs well with small time requirements. This could be improved by having just one set of nested loops instead of two. Although this algorithm produced good images, the mean squared error was relatively high for unknown reasons. The downsampled images appear to be worse in quality because when they are downsampled, the only data that is saved is every other row and column. This means that half of the data is deleted, which means that achieving the same image after downsampling and then upsampling is impossible. Different images have different mean squared errors evidently from calculations. This is because pictures with many repeated colours in the vicinity will have less error since it is easy for the estimation algorithm. This was a very interesting project and much knowledge of interpolation and image processing was gained.