## CMPEN/EE455:  Digital Image Processing I

## Computer Project # 1:

## Lab Introduction and Digital Image Processing

#### Naisong Chen, Andy Luo, William Yoshida

#### Date: 09/07/2018

* + 1. **Objectives**

In this project, we:

* get acquainted with project requirements for CMPEN/EE 455,
* learn how to use MATLAB for digital image processing, and
* study the effects of varying the spatial and gray-scale resolution in a digital image.
  + 1. **Methods**

1. **Spatial Resolution Downsampling**

**Relevant files:**

* main.m
* downsample2x.m
* nn\_interpolation

**Main class: main.m**

Flow: For each of the 3 desired output spatial resolutions (256x256, 128x128, and 32x32) run the downsample2x function on the original number a certain number of times. The number of times depends on the level of spatial downscaling to do. From 512x512 to 256x256 is a 2x downscale. From 512x512 to 128x128 is a 4 times downscale (2^2). From 512x512 to 32x32 is a 16x downscale (2^4). Next upscale the image back to the 512x512 resolution by using nn\_interpolation function. The number of times 2x downsampling was performed is the same number of times that 2x upscaling is needed to represent the downscaled image in the original spatial resolution. Finally the resulting images are written out to an output directory ‘output\_images’.

**Functions:**

downsample2x(f) - Downsample the spatial resolution of a grayscale image by a factor of 2. (e.g. takes a 512x512 image and downsamples to 256x256) Does so using the average of 2x2 regions in original image to determine corresponding pixel values in the output image.

Flow:

* For each pixel in the downsampled image:
  + traverse the corresponding 2x2 region in the original image
  + calculate the average of that region and set value in downsampled image

nn\_interpolation(f, s) - Upsample a given image using the nearest neighbor interpolation algorithm. Takes an input parameter that specifies how many times 2x upscale interpolation should happen. (e.g. s = 3 means 64x64 image gets upscaled to 512x512 image, a factor of 8).

Flow:

* Setup the upsampled image to be 2^s width and height of original image
* For number of times to upscale by factor of 2:
  + for each pixel in the upscaled image:
    - determine the corresponding “neighbor” pixel in the original image, and set the value. To determine neighbor pixel coordinate, divide new image pixel coordinates by 2 and round to nearest integer.

1. **Inverse Distance Interpolation**

**Relevant Functions**

* + inversedistance()
  + emptyarray()
  + invdistcalc()

inversedistance() - computes the inverse-distance interpolation given a desired spatial resolution for the final image. To run the code for part 2, then, we read in an image and call the inversedistance() function with parameters f, A, and B. Parameter f is the image to be interpolated, and the other parameters are the desired AxB resolution. The returned result is the image, invdist, the interpolated result with the specified resolution.

Flow: 1. Find size of f and how many iterations will be needed  
 2. For each iteration,  
 a. create and fill a 4x larger empty image with known pixel values,  
 b. iterate through all pixels to find missing values, i.e.  
 those needing an interpolated value calculation,  
 and call invdistcalc() for these

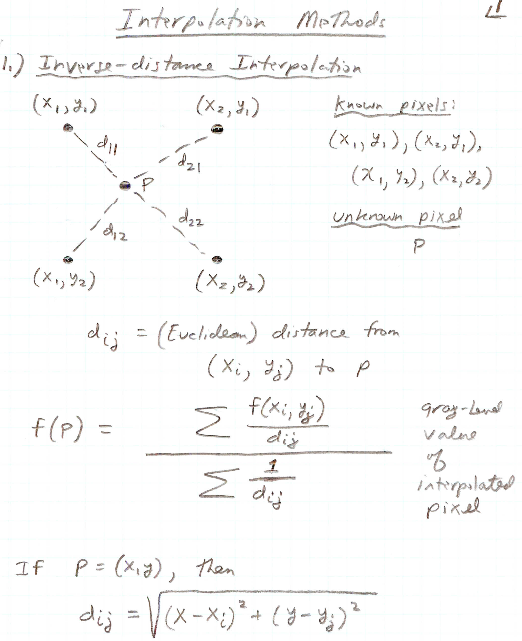
In step 1, we calculate iter, the number of times to conduct an iteration of our interpolation, which is dependent on the comparative size of f and invdist, i.e. how many times to expand the image. This is done because inversedistance() does not interpolate and upscale from a 32x32 image to a 512x512 image in one instant. From a 32x32, a 64x64 image is created, followed by a 128x128, 256x256, and so on up to 512x512. For a given iteration, then, a 4x larger empty image is created by calling the emptyarray() function (step 2a). When a pixel with no value is found in step 2b, invdistcalc() is called to calculated its value. This process is repeated until a final image is created with the desired resolution.

emptyarray() - creates an empty image given a desired spatial resolution. This function takes in parameters M and N, the desired resolution for the empty image. The returned result is the image, emptyimage. The purpose of this function is to create an empty image (an image filled with NaN’s), rather than an image filled with zeroes. This is done so inversedistance() can identify missing pixel values. If a zero-filled image was used instead, inversedistance() may have a harder time distinguishing missing values from real pixels with a value of 0.

Flow:

1. Cycle through the MxN array and fill with NaN's (Not a Number)

invdistcalc() - calculates the target pixel's value using inverse-distance interpolation. This function takes in parameters f, x, y, M, and N. Parameter f is our pre-interpolation image with missing pixel values. Parameters x and y are the coordinates for our target pixel, and M and N is the resolution of f. The returned result is the value, interpolval, the calculated value for our target pixel. For its inverse-distance calculation, invdistcalc() uses the pixels in the target’s 8-connected neighborhood. The following is the formula used for the calculation:



**Figure 1.** Inverse-distance Interpolation Formula

To find the distance between the target pixel, *P,* and an 8-connectivity neighboring pixel, the Euclidean distance formula, dij, as shown in Figure 1, was used. To calculate the value for the target pixel, we use the formula f(P) with (xi,yj) being the 8-connected neighbor.

Flow: 1. Cycle through all 8-neighborhood pixels, ignoring beyond-border and

other missing pixel values  
 2. For every valid pixel, calculate distance, numerator, and denominator

values for the inverse-distance formula

3. Calculate final inverse-distance interpolation value

For step 1, we iterate through all the 8-connected neighbors, including the target pixel itself. We then have to check for and ignore the target pixel, any pixels with missing values as well, and coordinates beyond the boundaries, i.e. non-existing pixels. For step 2 and 3, then, we only perform the formula calculation with the valid pixels.

1. **Gray-scale Resolution**

**Relevant files:**

* main.m
* grayscale\_downsample.m

**Main class: main.m**

Flow:

Calls the grayscale\_downsample function for the original input image and each of the bits values between 1 and 7 inclusive, saving the output grayscale downsampled files to the output directory ‘output\_images’.

**Functions:**

grayscale\_downsample(f, b) - downsamples the grayscale resolution to use a specified number of bits, b. This value is lower than the original number of bits for the image’s grayscale resolution.

Flow:

The first step will read the input image. Then we divided by the corresponding (gray\_bits) number of the desired value of bits per pixel. For instances, we want to get 7 bits/pixel, the corresponding number will be 2^1 which is 2. Thus, 6 bits/pixel will be 2^2 = 4, and so on. After division, we can get the desired ratio of the grayscale. For example. for 1 bits/pixel, we will be divided by 128 (2^7), the result give us the ratio between 0 to 2. Therefore, we multiply the results from the division by the same gray\_bits to get the gray level we want.

1. **Spatial and Gray-scale Resolution Combined Manipulation**

**Relevant files:**

* main.m
* downsample2x.m
* nn\_interpolation.m
* grayscale\_downsample.m

**Main class: main.m**

Flow: The portion of the main script responsible for this step first reads the input image and performs the downsample2x function on it, downsampling the 512x512 image into a 256x256 one. Next the nn\_interpolation function is run on the image to represent the downsampled as a 512x512 image, using nearest neighbor interpolation. Finally, the grayscale\_downsample function is called on the image to downsample the grayscale level to 6 bit representation.

* + 1. **Results**

Here is the original 512x512 grayscale “walkbridge.tif” image that was used for our

manipulations (Figure 2).



**Figure 2.** Original “walkbridge.tif” image (512x512) tested

***Running main.m script will generate all of the necessary images in the ‘output\_images’ directory.***

* + - 1. **Spatial Resolution Downsampling**



Figure 3. “walkbridge.tif” image downsampled to 256x256, then upsampled back to 512x512 using nearest neighbor interpolation



Figure 4. “walkbridge.tif” image downsampled to 128x128, then upsampled back to 512x512 using nearest neighbor interpolation

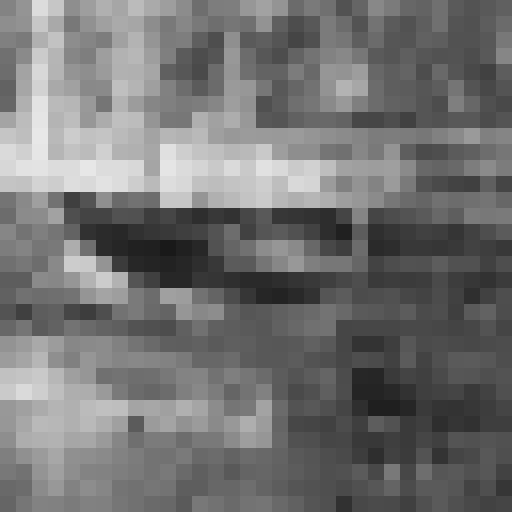


Figure 5. “walkbridge.tif” image downsampled to 32x32, then upsampled back to 512x512 using nearest neighbor interpolation

For every 2x reduction in spatial resolution, there is a great reduction in image detail. This is clearly evident progressing from Figure 3, to Figure 4, to Figure 5, where each one has much less detail than the one before, to the point where in Figure 5, you can clearly see each pixel. This is largely due to the fact that a 2x reduction in both width and height results in 4 times less total pixels in the image, and losing 4x as many pixels per 2x downscale quickly degrades the image quality, even with the full 8-bit grayscale representation. Although general scene structures such as the bridge posts are still distinguishable in the 256x256 (Figure 3) and 128x128 (Figure 4) images but not at all in the 32x32 image (Figure 5). At that point, the pixels correspond to a physical size wider than the size of the bridge posts, and therefore they become indistinguishable.

The nearest neighbor interpolation technique does not affect the apparent spatial resolution, even though the lower resolutions images are housed in a higher resolution. Nearest neighbor interpolation simply takes the nearest corresponding neighbor pixel in the original smaller image and assigns it to the pixel in the interpolated image. For example, nearest neighbor interpolation for a 2x upsample would simply split each pixel into four pixels with the same grayscale value, preserving even the square shape of the pixel. This type of direct downsampling has yielded the expected results.

* + - 1. **Inverse Distance Interpolation**



**Figure** 6**.** Final 512x512 interpolated result from downsampled 32x32 walkbridge image using inverse-distance method with 8-neighbour surrounding pixels

This is the final image inverse-distance interpolated from a 32x32 resolution to a 512x512. We see that it is no way nearly as detailed as Figure 2, the original image, though they have the same resolution; it is still considerably blurred as expected. This happens because it is deriving its level of detail from the 32x32 version. As the 32x32 version has less pixels, it holds less détail. To upsample, the values for the remaining pixels were estimated given the the nearby pixel values, which is why the level of détail does not return. What can be observed, however, is that the interpolated version is definitely smoother than the 32x32 version. This is due to having calculated pixel values, forming smoother gradients.

It can also be observed that there are still some pixels that stand out, which are either noticeably brighter or darker than those around them. This occurs because the original pixel values for the 32x32 version were kept; the surrounding pixels are the pixels that were calculated. These values were found by getting the average of a group of pixels, which is why they show a brighter or a darker intensity compared to the calculated pixels. This phenomenon was not expected, but in retrospect, it makes sense why it would occur. Perhaps this could be avoided by replacing the original pixel values by a new calculated value, this time using the neighboring pixels that were calculated themselves. Hopefully, this might smooth out the image further.

* + - 1. **Gray-scale Resolution**



Figure 7. “walkbridge.tif” image downscaled to use 1 bit to represent grayscale levels



Figure 8.. “walkbridge.tif” image downscaled to use 2 bits to represent grayscale levels



Figure 9. “walkbridge.tif” image downscaled to use 3 bits to represent grayscale levels



Figure 10. “walkbridge.tif” image downscaled to use 4 bits to represent grayscale levels



Figure 11. “walkbridge.tif” image downscaled to use 5 bits to represent grayscale levels



Figure 12. “walkbridge.tif” image downscaled to use 6 bits to represent grayscale levels



Figure 13. “walkbridge.tif” image downscaled to use 7 bits to represent grayscale levels

With less bits representing the grayscale levels of an image, there is also a decrease in perceived quality of the image, however in a slightly different way than spatial resolution downsampling. The less bits there are, the more loss in intensity detail, not necessarily scene structure detail. Going from 8 bit (Figure 13) to 7 bit (Figure 12) representation has a negligible effect on image quality, almost all the original detail in both the foreground and background are present. However going from 3 bit (Figure 9) to 2 bit (Figure 8) representation has a huge impact on image quality, with entire regions of of similar but still different grayscale levels in the 3 bit representation becoming a single mono-level region in the 2 bit representation. This is because each decline in a bit corresponds to halving the number of grayscale levels, and there are already so few levels at 3 bits (relative to the number of levels the human eye can perceive).

This behavior is as expected and our results properly demonstrate the loss in detail that comes with grayscale downsampling, even if scene structure is more preserved than spatial resolution downsampling. Regions that have similar grayscale levels become a singular grayscale level after downsampling, even if the region boundary is somewhat well preserved.

* + - 1. **Spatial and Gray-scale Resolution Combined Manipulation**



**Figure 14.** 512x512 image after original image had spatial resolution reduced to 256x256 and grayscale resolution to 6 bits/pixel

Due to the resolutions being lowered, we see that the resulting image begins to form single-intensity components or regions where previously detailed regions that had a nearly constant intensity were before. This is expected since after resolution reduction, regions with near-constant intensity would become constant. This happened, for instance, with the water, which did have more détail, but since water regions tend to have approximately the same intensity, distinct single-intensity regions arise.

After conducting both operations on the spatial and grayscale resolution, some artifacts begin to appear on the image. This happens frequently in images after several processing operations have been conducted, particularly if resolutions were reduced. Due to the nature of how the sunlight falls, hits, and covers objects in the image, a lower resolution is not capable of retaining all the level of détail caused by the sunlight the original image had. For instance, behind one of the single tree trunks on the background at the end of the bridge we begin to see an artifact that may resemble the figure of a man. If we look at the original image, Figure 2, we see that, in reality, the man, is simply a bunch of leaves covered in sunlight. The difference in intensity to the rest of the leaves caused the artifact to appear after the resolution was reduced.

* + 1. **Conclusions**

Spatial and grayscale resolution play a large role in the perceived quality of an image. Decreasing either will decrease the perceived image quality, however depending on how much the resolution changes and how high of resolution the original image was at, this change may be drastic or imperceptible, or somewhere in between. This supports the fact that human eyes have a limit to detail perception, both in the spatial sense, and the intensity sense. Halving the grayscale or spatial resolution of an image that already has high grayscale or spatial resolution would not affect the perceived image quality much, or at all. Additionally, different methods of upscaling an images have varying levels of perceived quality increase. For example, nearest neighbor interpolation has no visible quality improvement, but inverse-distance interpolation has some detail reconstruction. All of these principles are extremely important in an image-heavy, media driven world, where there will always be a tradeoff between perceived quality and storage size.