CSCI-576: Assignment 1

Zhanshan Zhang, id = 7527742899

Explanation

The implementation adds a new method called `resampleImage()` to create a new resampled image. Based on the input parameters, there are 4 methods to do the corresponding resampling.

Command line format:

```
java ImageDisplay [imgPath] [width] [height] [resamplingMethod] [targetImgSize]
// example
java ImageDisplay ../hw1_1_high_res.rgb 4000 3000 1 O1
java ImageDisplay ../hw1_1_high_res.rgb 4000 3000 2 O1
java ImageDisplay ../hw1_1_low_res.rgb 400 300 3 O1
java ImageDisplay ../hw1_1_low_res.rgb 400 300 4 O1
java ImageDisplay ../hw1_1_high_res.rgb 4000 3000 5 O1 // Not required
```

There are 5 methods for `[resamplingMethod]`:

`resamplingMethod = 1`: down-sample, specific sampling

`resamplingMethod = 2`: down-sample, average smoothing

`resamplingMethod = 3`: up-sample, nearest neighbor

`resamplingMethod = 4`: up-sample, bilinear Interpolation

`resamplingMethod = 5`: down-sample, Pixel Aspect Ratio (PAR) // Not required

Method 1

down-sample, specific sampling

This method find the pixel in the original image proportionally and fill it to the target image.

targetImage(x, y) = originalImage(x * oldW / newW, y * oldH / newH)

Output: This outputs a reduced image.

Method 2

down-sample, average smoothing

This method find the blocks of pixels in the original image, calculate their average value, and fill it to the target image.

Output: This outputs a reduced image, but the image is smoother.

Method 3

up-sample, nearest neighbor

This method find the pixel in the original image proportionally and fill it to the target image. Adjacent pixels in the target image might have the same value because they are mapped into the same pixel in the original image.

Output: This outputs a enlarged image, but the image is unclear and jagged.

Method 4

up-sample, bilinear Interpolation

This method do bilinear interpolation for pixels in original image to calculate the pixels in the target image.

- For each pixel (x, y) in the target image.
- The pixel is mapped into position between (x1, y1), (x2, y2), (x3, y3), (x4, y4) in the original image.
- xDiff = x * oldW / newW - x1, yDiff = y * oldH / newH - y1
- The value of the pixel (x, y) in the target image follow this formula:
- (x, y) = (x1, y1) * (1 - xDiff)(1 - yDiff) + (x2, y1) * (1 - xDiff) * yDiff + (x1, y2) * xDiff * (1 - yDiff) + (x2, y2) * xDiff * yDiff

Output: This outputs a enlarged image. Due to bilinear interpolation, the image is clearer than the one in Method 3.

Discussion

Pixel Aspect Ratio (PAR) changes while down sampling

Method: Maintain pixels at the center of the image without any stretching, and increase the stretching nonlinearly towards the periphery.

This is implemented by using a non-liner function. Here for simplicity, I choose a quadratic function which is approximately linear at the center of the image, and non-linear at the periphery.

Note that the modified image is complete (just stretched). You can see the complete information in the four corners of the image.

Outputs:

Pixel Aspect Ratio, O1:

```
java ImageDisplay ../hw1_1_high_res.rgb 4000 3000 5 O1
```

Pixel Aspect Ratio, O2:

```
java ImageDisplay ../hw1_1_high_res.rgb 4000 3000 5 O2
```

You can also see all the original images on Github: https://github.com/seawolf1024/CSCI-576

Seam Carving as a solution

Comment on the results:

The smooth areas of the image are removed first, while the areas with significant abrupt changes are retained. In this way, the main object is retained, and it's with the same proportions as in the original image.

Pros:

(1) Content Preservation: Seam carving could preserve important image content. Objects of interest (e.g. people, objects) tend to remain undistorted because the algorithm removes less significant pixels.

(2) Avoiding Distortion: Seam carving avoids the stretching or squashing of objects.

Cons:

(1) Performance in high-frequency Areas: See **Where it Struggles** below.

(2) Time complexity: Seam carving requires generating the energy map, finding seams, and removing them. This can take a long time for large images.

Where do you think the method performs well or does not perform well?

Where it Performs Well: It performs well in images wiht large homogeneous regions (e.g. skies, fields), as it can safely remove seams in these areas without affecting the overall composition.

Where it Struggles: The method struggles with complex or densely populated images (e.g. those with multiple people/objects, high-frequency textures). In such cases, removing seams can lead to visual distortions.

Outputs:

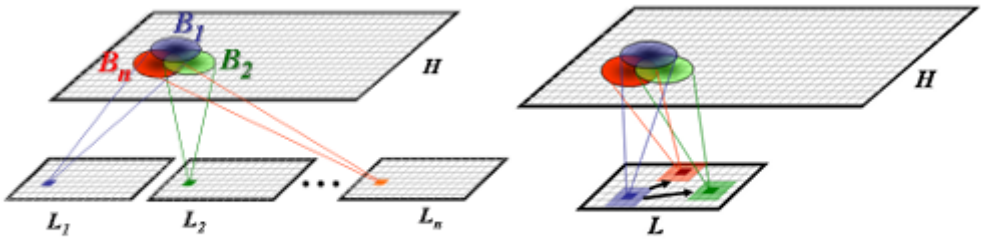
Here I deduce the width and height of each image both by 50:

Up sampling image quality

Explanation:

As from the paper, the idea is to use patches from the same locations of multiple low-resolution images (with subpixel misalignment) to fit the corresponding areas in the high-resolution image.

Key idea: For each patch in the low-resolution image, use NN to find similar patches, then map the high-resolution receptive field, and establish simultaneous equations through the overlapping parts.



(a) Classical Multi-Image SR (b) Single-Image Multi-Patch SR

Outputs: