WIKIPEDIA

Directional Cubic Convolution Interpolation

Directional Cubic Convolution Interpolation (DCCI) is an edge-directed image scaling algorithm created by Dengwen Zhou and Xiaoliu Shen.[1]

By taking into account the edges in an image, this scaling algorithm reduces artifacts common to other image scaling algorithms. For example, staircase artifacts on diagonal lines and curves are eliminated.

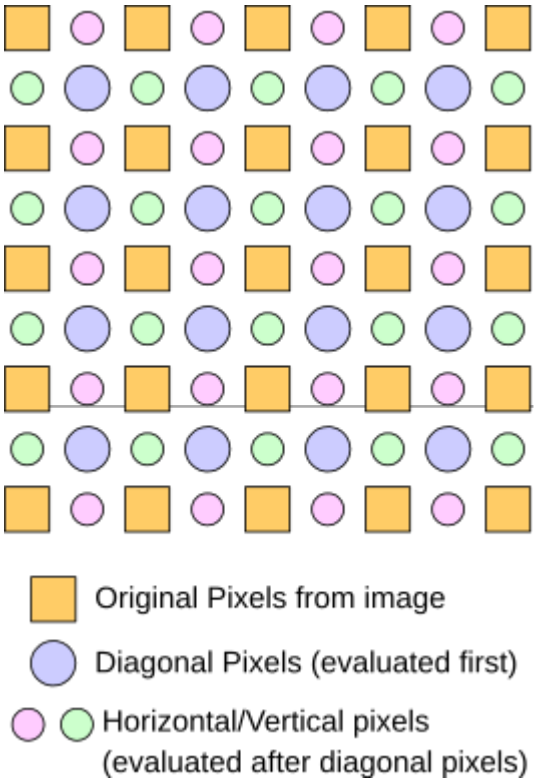
The algorithm resizes an image to 2x its original dimensions, minus 1.

Contents

The algorithm

The algorithm works in three main steps:

1. Copy the original pixels to the output image, with gaps between the pixels.
2. Calculate the pixels for the diagonal gaps.
3. Calculate the pixels for the remaining horizontal and vertical gaps.



Calculating pixels in diagonal gaps

Evaluation of diagonal pixels is done on the original image data in a 4×4 region, with the new pixel that is being calculated in the center, in the gap between the original pixels. This can also be thought of as the 7×7 region in the enlarged image centered on the new pixel to calculate, and the original pixels have already been copied.

The algorithm decides one of three cases:

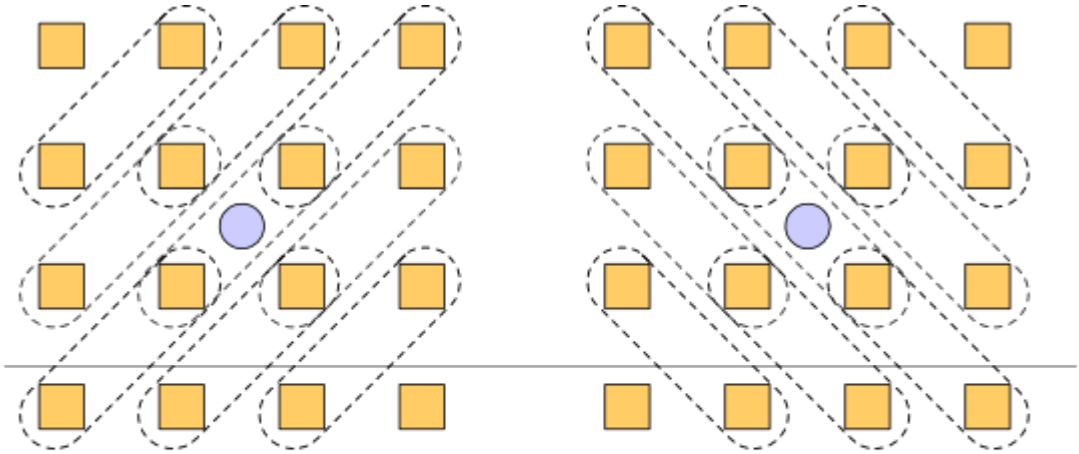
- Edge in up-right direction — interpolates along down-right direction.
- Edge in down-right direction — interpolates along up-right direction.
- Smooth area — interpolates in both directions, then multiples the values by weights.

Calculating diagonal edge strength

Let d1 be the sum of edges in the up-right direction, and d2 be the sum of edges in the down-right direction.

To calculate d1, take the sum of abs(P(X, Y) - P(X - 1, Y + 1)), in the region of X = 1 to 3, and Y = 0 to 2.

To calculate d2, take the sum of abs(P(X, Y) - P(X + 1, Y + 1)), in the region of X = 0 to 2, and Y = 0 to 2.



Up-Right direction: (d1)
For the three right-side pixels in the first three rows, compute the absolute difference | P(X, Y) - P(X - 1, Y + 1) | then sum all 9 values together.

Down-Right direction: (d2)
For the three left-side pixels in the first three rows, compute the absolute difference | P(X, Y) - P(X + 1, Y + 1) | then sum all 9 values together.

Interpolating pixels

If (1 + d1) / (1 + d2) > 1.15, then you have an edge in the up-right direction.

If (1 + d2) / (1 + d1) > 1.15, then you have an edge in the down-right direction.

Otherwise, you are in the smooth area.

To avoid division and floating-point operations, this can also be expressed as 100 * (1 + d1) > 115 * (1 + d2), and 100 * (1 + d2) > 115 * (1 + d1).

Up-right edge

For an edge in the up-right direction, we want to interpolate in the down-right direction.

Output pixel = (-1 * P(0, 0) + 9 * P(1, 1) + 9 * P(2, 2) - 1 * P(3, 3)) / 16

The pixel value will need to be forced to the valid range of pixel values (usually 0 to 255).

Down-right edge

For an edge in the down-right direction, we want to interpolate in the up-right direction.

Output pixel = (-1 * P(3, 0) + 9 * P(2, 1) + 9 * P(1, 2) - 1 * P(0, 3)) / 16

The pixel value will need to be forced to the valid range of pixel values (usually 0 to 255).

Smooth area

In the smooth area, edge strength from up-right will contribute to the down-right sampled pixel, and edge strength from down-right will contribute to the up-right sampled pixel.

w1 = 1 / (1 + d1 ^ 5)

w2 = 1 / (1 + d2 ^ 5)

weight1 = w1 / (w1 + w2)

weight2 = w2 / (w1 + w2)

DownRightPixel = (-1 * P(0, 0) + 9 * P(1, 1) + 9 * P(2, 2) - 1 * P(3, 3)) / 16

UpRightPixel = (-1 * P(3, 0) + 9 * P(2, 1) + 9 * P(1, 2) - 1 * P(0, 3)) / 16

Output Pixel = DownRightPixel * weight1 + UpRightPixel * weight2

The pixel value will need to be forced to the valid range of pixel values (usually 0 to 255).

Calculating remaining pixels

Evaluation of remaining pixels is done on the scaled image data in a 7×7 region, with the new pixel that is being calculated in the center. These calculation either depend on the original pixels of the image, or a diagonal pixel calculated in the previous step.

The algorithm decides one of three cases:

- Edge in horizontal direction — interpolates along vertical direction.
- Edge in vertical direction — interpolates along horizontal direction.
- Smooth area — interpolates in both directions, then multiples the values by weights.

Calculating horizontal/vertical edge strength

Let d1 be the sum of edges in the horizontal direction, and d2 be the sum of edges in the vertical direction.

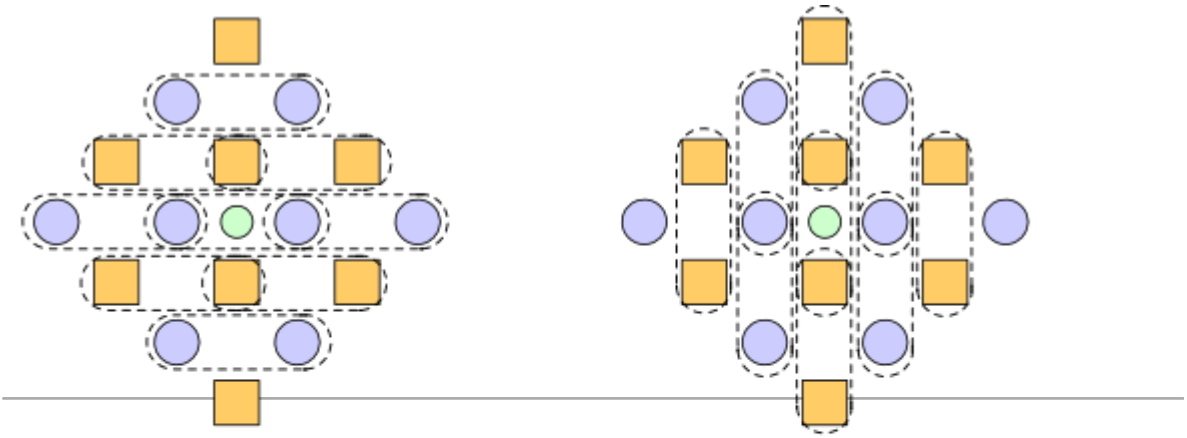
Consider a 7×7 diamond-shaped region centered on the pixel to calculate, using only pixel values from the original, and pixel values added from the diagonal direction.

To calculate d1, take the sum of the absolute differences of the horizontal edges, sampling these pixel values:

```
| P(X+1, Y-2) - P(X-1, Y-2) | +
| P(X+2, Y-1) - P(X, Y-1) | + | P(X, Y-1) - P(X-2, Y-1) | +
| P(X+3, Y) - P(X+1, Y) | + | P(X+1, Y) - P(X-1, Y) | + | P(X-1, Y) - P(X-3, Y) | +
| P(X+2, Y+1) - P(X, Y+1) | + | P(X, Y+1) - P(X-2, Y+1) | +
| P(X+1, Y+2) - P(X-1, Y+2) |
```

To calculate d2, take the sum of the absolute differences of the vertical edges, sampling these pixel values:

```
| P(X-2, Y+1) - P(X-2, Y-1) | +
| P(X-1, Y+2) - P(X-1, Y) | + | P(X-1, Y) - P(X-1, Y-2) | +
| P(X, Y+3) - P(X, Y+1) | + | P(X, Y+1) - P(X, Y-1) | + | P(X, Y-1) - P(X, Y-3) | +
| P(X+1, Y+2) - P(X+1, Y) | + | P(X+1, Y) - P(X+1, Y-2) | +
| P(X+2, Y+1) - P(X+2, Y-1) |
```



Horizontal direction: (d1)
With X, Y in the center of the image:
Compute the sum of absolute differences:
| P(X+1, Y-2) - P(X-1, Y-2) | +
| P(X+2, Y-1) - P(X, Y-1) | + | P(X, Y-1) - P(X-2, Y-1) | +
| P(X+3, Y) - P(X+1, Y) | + | P(X+1, Y) - P(X-1, Y) | +
| P(X-1, Y) - P(X-3, Y) | +
| P(X+2, Y+1) - P(X, Y+1) | + | P(X, Y+1) - (P(X-2, Y+1) | +
| P(X+1, Y+2) - P(X-1, Y+2) |

Vertical direction: (d2)
With X, Y in the center of the image:
Compute the sum of absolute differences:
| P(X-2, Y+1) - P(X-2, Y-1) | +
| P(X-1, Y+2) - P(X-1, Y) | + | P(X-1, Y) - P(X-1, Y-2) | +
| P(X, Y+3) - P(X, Y+1) | + | P(X, Y+1) - P(X, Y-1) | +
| P(X, Y-1) - P(X, Y-3) | +
| P(X+1, Y+2) - P(X+1, Y) | + | P(X+1, Y) - (P(X+1, Y-2) | +
| P(X+2, Y+1) - P(X+2, Y-1) |

Interpolating pixels

If (1 + d1) / (1 + d2) > 1.15, then you have an edge in the horizontal direction.

If (1 + d2) / (1 + d1) > 1.15, then you have an edge in the vertical direction.

Otherwise, you are in the smooth area.

To avoid division floating-point operations, this can also be expressed as 100 * (1 + d1) > 115 * (1 + d2), and 100 * (1 + d2) > 115 * (1 + d1).

Horizontal edge

For a horizontal edge, we want to interpolate in the vertical direction, using only the column centered at the pixel.

Output pixel = (-1 * P(X, Y - 3) + 9 * P(X, Y - 1) + 9 * P(X, Y + 1) - 1 * P(X, Y + 3)) / 16

The pixel value will need to be forced to the valid range of pixel values (usually 0 to 255).

Vertical edge

For a vertical edge, we want to interpolate in the horizontal direction, using only the row centered at the pixel.

Output pixel = (-1 * P(X - 3, Y) + 9 * P(X - 1, Y) + 9 * P(X + 1, Y) - 1 * P(X + 3, Y)) / 16

The pixel value will need to be forced to the valid range of pixel values (usually 0 to 255).

Smooth area

In the smooth area, horizontal edge strength will contribute to the weight for the vertically sampled pixel, and vertical edge strength will contribute to the weight for the horizontally sampled pixel.

w1 = 1 / (1 + d1 ^ 5)

w2 = 1 / (1 + d2 ^ 5)

weight1 = w1 / (w1 + w2)

weight2 = w2 / (w1 + w2)

HorizontalPixel = (-1 * P(X - 3, Y) + 9 * P(X - 1, Y) + 9 * P(X + 1, Y) - 1 * P(X + 3, Y)) / 16

VerticalPixel = (-1 * P(X, Y - 3) + 9 * P(X, Y - 1) + 9 * P(X, Y + 1) - 1 * P(X, Y + 3)) / 16

Output Pixel = VerticalPixel * weight1 + HorizontalPixel * weight2

The pixel value will need to be forced to the valid range of pixel values (usually 0 to 255).

Not specified

Boundary pixels

The algorithm does not define what to do when sampling boundary areas outside of the image. Possible things to do include replicating the boundary pixel, wrapping pixels from the other side of the image, wrapping the same side of the image in reverse, or using a particular border color value.

Color images

Color images aren't specified by the algorithm, however, you can sum all RGB component differences when calculating edge strength, and use all RGB components when interpolating the pixels. Or you could split to YCbCr, process only the luma component and stretch the chroma using a different algorithm.

See also

- Bilinear interpolation
- Bicubic interpolation
- Lanczos resampling

References

1. Dengwen Zhou; Xiaoliu Shen. "Image Zooming Using Directional Cubic Convolution Interpolation" (http://www.mathworks.com/matlabcentral/fileexchange/38570-image-zooming-using-directional-cubic-convolution-interpolation). Retrieved 13 September 2015.

/w/index.php?title=Directional_Cubic_Convolution_Interpolation&oldid=797783472"