**IMAGE GEOMETRIC TRANSFORMATION**

**GEOMETRIC TRANSFORMATION**

The operations which modify the spatial relationship between the pixels in an image are called Geometric transformation. These transformations mainly modify the position of pixels rather the intensities. Geometric transformations require 2 steps.

1. **Spatial Mapping**

Spatial Mapping is transforming a point (x,y) in an image to the new coordinate system (u,v) by applying some functions. During this transformation, we may get points which are not integers giving a need for the 2nd step Interpolation

u = f1(x, y)

A picture containing sky, object

Description generated with high confidencev = f2(x, y)

(u,v)

(x,y)

<https://www.cis.rit.edu/class/simg782/lectures/lecture_02/lec782_05_02.pdf>

1. **Interpolation**

A process of estimating unknown values using the know values in called Interpolation.

**Applications**

Geometric transformation has a wide variety of application. Some of them are

1. Aligning images taken at different times is a structures way to make the things more organized.
2. They are used to correct images for lens distortion and correct effects of camera orientation.
3. They also play a key role in Image Morphing or creating special effects.
4. They are used in image pre-processing steps in applications such as document understanding, where the scanned image may be mis-aligned.

**IMPLEMENTATION**

**GUI**

The GUI is implemented using the Tkinter Package of Python.

**Scaling**

Scaling performs geometric transformation which can be used to shrink or zoom the size of an image. It changes the size of the image by changing the no. of pixels. Zooming is used to make the details of the image more clear and visible. Shrinking is mostly used to create thumbnail images.

To implement this, we take the input from the user, the factor which they want to scale the image in x and y directions along the interpolation technique they prefer to use. We have 4 different interpolations techniques namely Nearest Neighbor, Bilinear, Bicubic, Lanczos4.

Using the x and y factors provided by the user, a new image containing all 0s is created.

Width of the new image = int(original width \* x-scale factor)

Height of the new image = int(original height \* y-scale factor)

Now we traverse every cell in the new image and find the mapping in the original image.

Row mapping = int(current row / y-scale factor)

Column mapping = int(current column / x-scale factor)

This mapping is used to identify the pixel in the original image and with the help of interpolation technique, we find the value for the current cell.

**Interpolation Techniques**

**Nearest Neighbor Interpolation**

In this interpolation method, the position of a pixel in the resultant image (expanded or shrunk) is converted back into the original image. The intensity of this pixel is set to the intensity of the pixel nearest to it. This method is simple but discontinuous, thus has no regularity.

**Bilinear Interpolation**

Bilinear interpolation unlike nearest neighbor considers the closest 2x2 neighborhood pixel values surrounding the unknown pixel's location. First the linear interpolation is done along rows and then interpolation is done along the resultant column.

The linear interpolation in one direction is given by the expression:

𝐼 = 𝐼1 + (𝐼2−𝐼1) \* (𝑥 − 𝑥1/ 𝑥2 − 𝑥1)

Where x is a point in between two points x1 and x2 and I1 and I2 are their intensities respectively.

**Bicubic Interpolation**

Bicubic interpolation solves for the value at the unknown pixel's location by considering the 16-pixel values surrounding the interpolation region. First the horizontal cubic interpolation is done and then the vertical cubic interpolation. For the horizontal interpolation portion of this algorithm, a cubic must be defined for each row of the 4x4 pixel region.

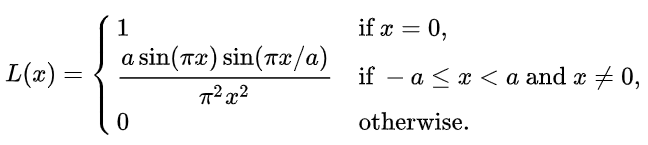
The values are inserted into the cubic and solved for each of the 4 pixels in the row. These 4 linear equations can be used to solve for the coefficients A, B, C and D.

After solving all the row interpolations, a vertical cubic interpolation is done through the row interpolation points. Using the same technique as with the row interpolations, the value of y is plugged into the cubic for each of the known row offsets and the coefficients are calculated. Thus, finding the intensity at the unknown pixel’s location.

**Lanczos 4 Interpolation**

This method is based on the 4-lobed Lanczos window function as the interpolation function, hence termed as Lanczos order four interpolation. It is to be used as a low-pass filter to smoothly interpolate the value of a digital signal between its samples. It uses source image intensities at 64 pixels nearest to unknown pixel.

The interpolated at value *x* is obtained by the discrete convolution of those samples with the Lanczos kernel along each row and then along the resultant column. The Lanczoz kernel (L(x)) is a normalized sinc function multiplied by sinc window.



**Time Complexity Evaluation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Original size (hxw) | New Size (hxw) | Interpolation | Time (Built-in) sec | Time (Custom) sec |
| 512x512 | 256x256 | Nearest Neighbor | 0.001 | 0.20 |
| 512x512 | 256x256 | Bilinear | 0.002 | 1.99 |
| 512x512 | 256x256 | Cubic | 0.002 | 4.08 |
| 512x512 | 256x256 | Lanczos4 | 0.01 | 14.20 |
| 512x512 | 512x768 | Nearest Neighbor | 0.108 | 1.16 |
| 512x512 | 512x768 | Bilinear | 0.005 | 11.84 |
| 512x512 | 512x768 | Cubic | 0.007 | 22.25 |
| 512x512 | 512x768 | Lanczos4 | 0.03 | 150.53 |
| 128x128 | 512x512 | Nearest Neighbor | 0.004 | 0.7966 |
| 128x128 | 512x512 | Bilinear | 0.012 | 7.2338 |
| 128x128 | 512x512 | Cubic | 0.004 | 33.3550 |
| 128x128 | 512x512 | Lanczos4 | 0.015 | 131.3345 |
| 128x128 | 1024x1024 | Nearest Neighbor | 0.007 | 3.2336 |
| 128x128 | 1024x1024 | Bilinear | 0.007 | 29.9864 |
| 128x128 | 1024x1024 | Cubic | 0.004 | 135.9492 |
| 128x128 | 1024x1024 | Lanczos4 | 0.019 | 474.3521 |
| 128x128 | 1280x1280 | Nearest Neighbor | 0.01 | 4.2490 |
| 128x128 | 1280x1280 | Bilinear | 0.008 | 45.1465 |
| 128x128 | 1280x1280 | Cubic | 0.007 | 211.6101 |
| 128x128 | 1280x1280 | Lanczos4 | 0.02 | 884.8461 |

Computational time in seconds using Intel® Core™ i5 [CPU@2.50](mailto:CPU@2.50)GHz and RAM of 8GB.

With the custom functions, Nearest Neighbor and Bilinear are faster compared to the Cubic and Lanczos4. When the size of the image increases, the time taken for Scaling increases. The built-in functions perform faster about 1400 times in case of Lanczos4 shrinking.

When we shrink or zoom the image by a very high factor, then the quality of the image deteriorates.

**Image Quality Evaluation:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Interpolation Type** | **Feeling** | **Edges** | **overall** |
| Nearest Neighbor | mosaic or jagged effect | jagged | worst |
| Bilinear | blur, not sharp | blur | poor |
| Bicubic | fuzzy, sharp | more clear edges | better |
| Lanczos 4 | fuzzy, sharp  (similar to bicubic) | ringing | better |