Project Report on

Image Interpolation Algorithms – Comparative Study

Submitted by:

Aneesh Panchal - 2K20/MC/21

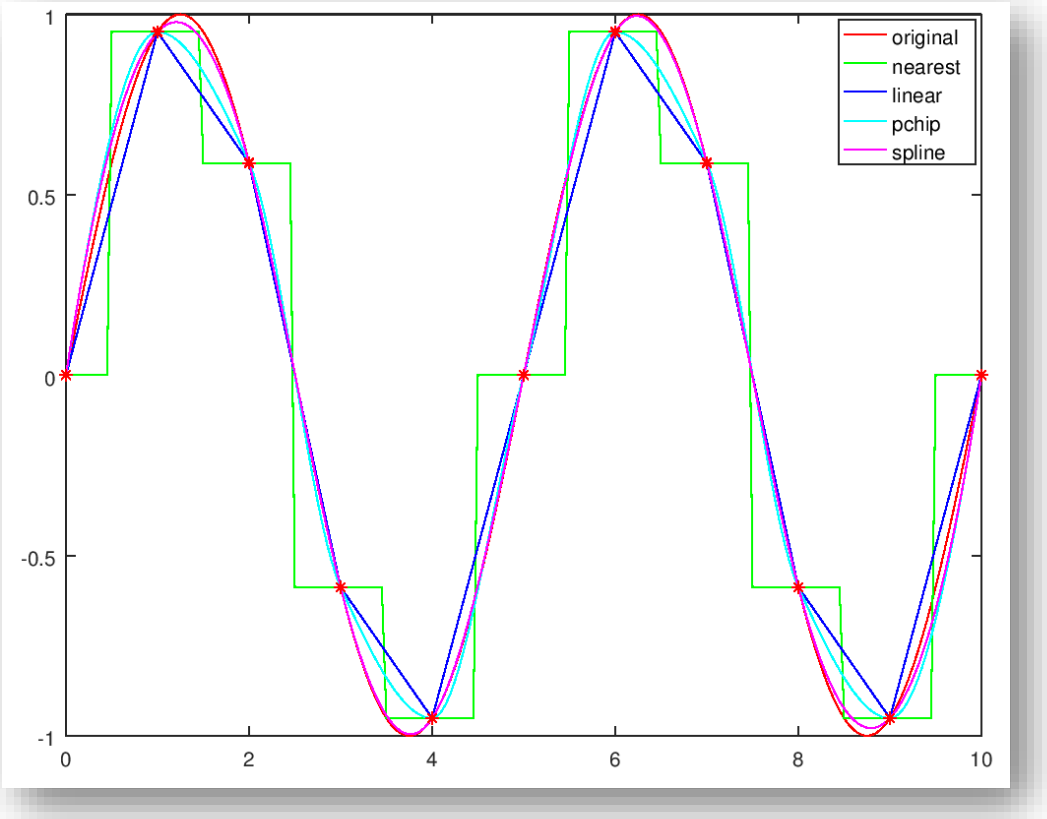
Ayushi sagar - 2K20/MC/35

Submitted to:

Prof. Ved Prakash Kaushik

Department of Applied Mathematics

Delhi Technological University



Certificate

I hereby certify that the project dissertation titled “Image Interpolation Algorithms - Comparative Study” which is submitted by Aneesh Panchal (2K20/MC/21) and Ayushi Sagar (2K20/MC/35) of Mathematics and Computing Department, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology, is a record of the project work carried out by the students. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this university or elsewhere.

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

Acknowledgement

We, Aneesh Panchal(2K20/MC/21) and Ayushi Sagar (2K20/MC/35) would like to express our special thanks of gratitude to Prof. Ved Prakash Kaushik, Department of Applied Mathematics, Delhi Technological University for his able guidance and support in completing our project report.

We came to know about many new things, we are really thankful to him. We would also like to thank our classmates who have also helped whenever we were stuck at some point.

Thanking You

Aneesh Panchal (2K20/MC/21)
Ayushi Sagar (2K20/MC/35)

Introduction

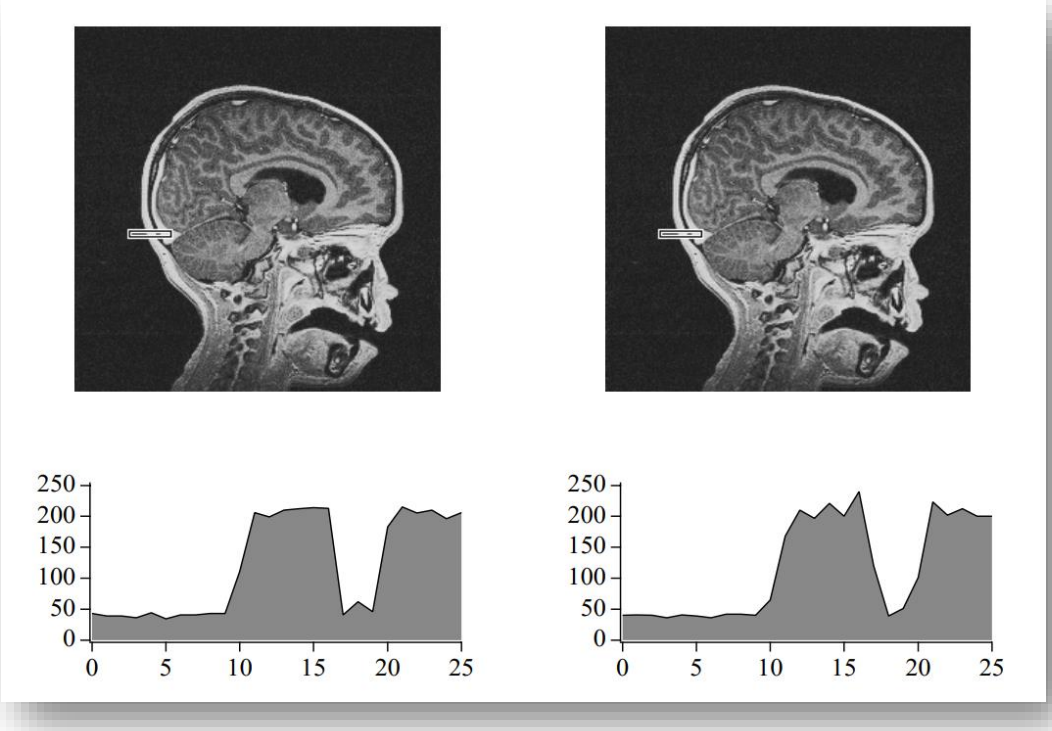
If the values of P(x) and/or its specific order derivatives coincide with those of f(x) and/or its same order derivatives at one or more tabular points, the polynomial is called an Interpolating polynomial. The following are some interpolation hypotheses:

1. The presented data (tabular points) are defined in a consistent manner.
2. The data value of the underlying continuous function must be computed at any moment.
3. At the Tabular points, the underlying continuous function evaluates to the same value as the data.

We get the interpolating polynomial of degree atmost (n-1) for n supplied locations (given points). The number of points is inversely proportional to the amount of mistake (error).

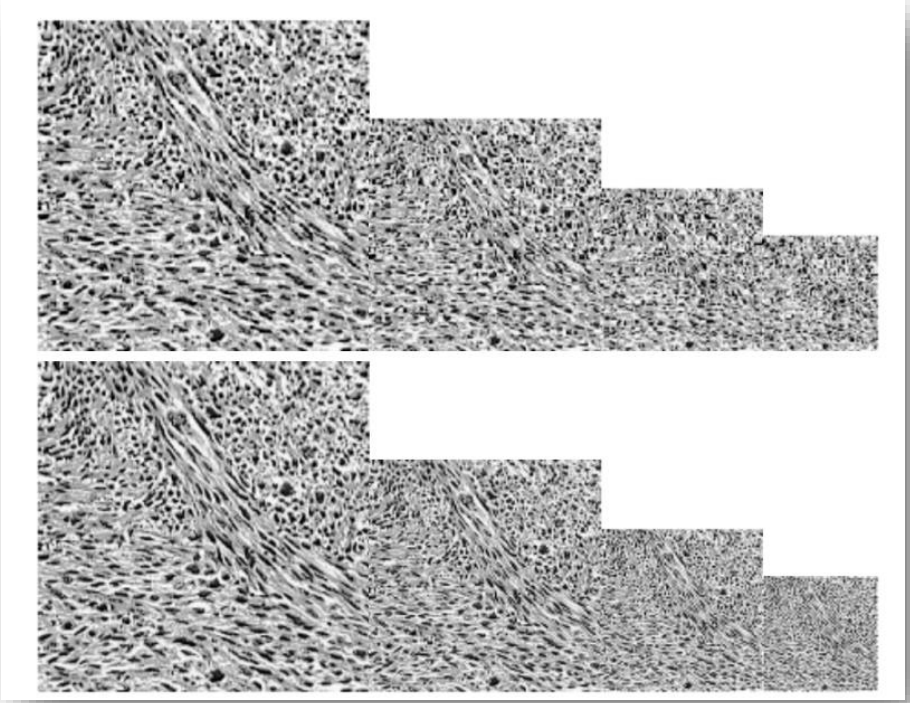
Applications of Interpolation:

1. The most obvious biomedical applications where interpolation is useful are those in which the goal is to change the sampling rate of pixels (picture elements) or voxels (volume elements). When an acquisition equipment, such as a scanner, has a non-homogeneous resolution, such as a fine within-slice resolution and a coarse across-slice resolution, this technique is called rescaling.

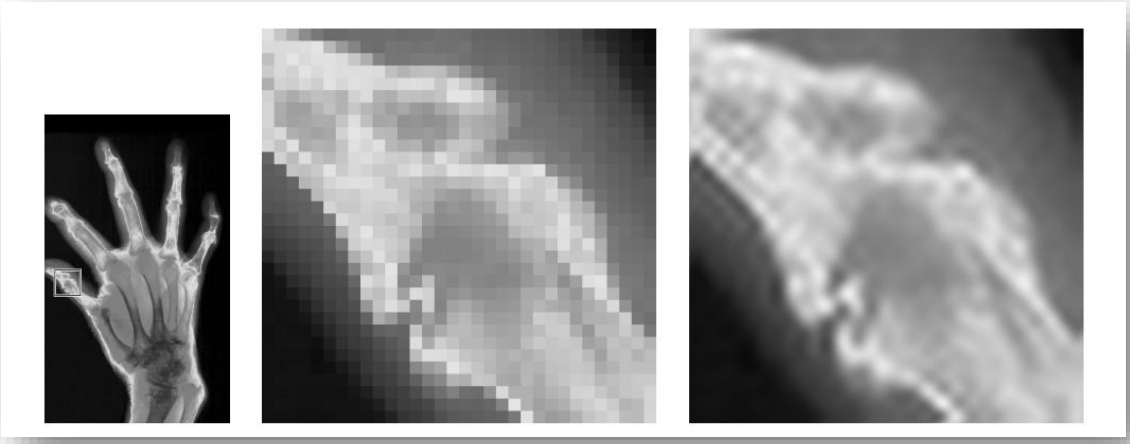


2. Interpolation's importance can also be seen in more advanced visualisation scenarios, such as volume rendering. It is normal practise to apply a texture to the facets that make up the displayed object there.

3. Images are a more common application of interpolation (as opposed to volumes). A clinician may want to examine an image at a coarse scale while also studying some detail at a fine scale. Interpolation procedures such as zooming in and out are helpful in this regard.



4. In biomedical imaging, when a picture must be translated, rotated, scaled, warped, or otherwise deformed before it can be compared to a reference image or atlas.



5. Tomographic reconstruction by (filtered) back-projection is an example of an algorithm that relies on interpolation to function effectively.

Image Interpolation

Image interpolation is a word that describes how images are processed (image scaling, image resampling and image resize).

Image interpolation algorithms are a phrase used to describe ways for expanding the number of pixels in an image.

Image interpolation for several purposes:

1. Change the size of the image by zooming in and out.
2. Image sharpness adjustment
3. Changing the image's blurriness, and so on...

Python Code (.py):

```
import os
import sys
import numpy as np
from scipy.interpolate import griddata
import matplotlib.pyplot as plt
from PIL import Image

def make_interpolated_image(nsamples):
    ix = np.random.randint(im.shape[1], size=nsamples)
    iy = np.random.randint(im.shape[0], size=nsamples)
    samples = im[iy,ix]
    int_im = griddata((iy, ix), samples, (Y, X))
    return int_im

img_name = sys.argv[1]
im = Image.open("CR.jpg")
im = np.array(im.convert('L'))

nx, ny = im.shape[1], im.shape[0]
X, Y = np.meshgrid(np.arange(0, nx, 1), np.arange(0, ny, 1))

nrows, ncols = 2, 2
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(6,4), dpi=100)
if nx < ny:
    w, h = fig.get_figwidth(), fig.get_figheight()
    fig.set_figwidth(h), fig.set_figheight(w)

get_indices = lambda i: (i // nrows, i % ncols)

for i in range(4):
    nsamples = 10**(i+2)
    axes = ax[get_indices(i)]
    axes.imshow(make_interpolated_image(nsamples),
                cmap=plt.get_cmap('Greys_r'))
    axes.set_xticks([])
    axes.set_yticks([])
    axes.set_title('n = {0:d}'.format(nsamples))
filestem = os.path.splitext(os.path.basename(img_name))[0]
```

Results:

As we can clearly see here that,

More number of points give clearer image as we get less error and higher degree interpolating polynomial from more number of points.

- From **100 points** we get interpolating polynomial of **degree 99**, hence we get very blurry image from which we can't conclude anything.
- From **1000 points** we get interpolating polynomial of **degree 999**, hence we get some idea that image is of some person but we can't conclude that whose image is given to us.
- From **10000 points** we get interpolating polynomial of **degree 9999**, hence we get to know that this image is of Cristiano Ronaldo but the image is still blurry.
- From **100000 points** we get interpolating polynomial of **degree 99999**, hence we get a clear image of Cristiano Ronaldo.

Hence, from above all we conclude that **more number of points** results in **clearer image** and **less error**.

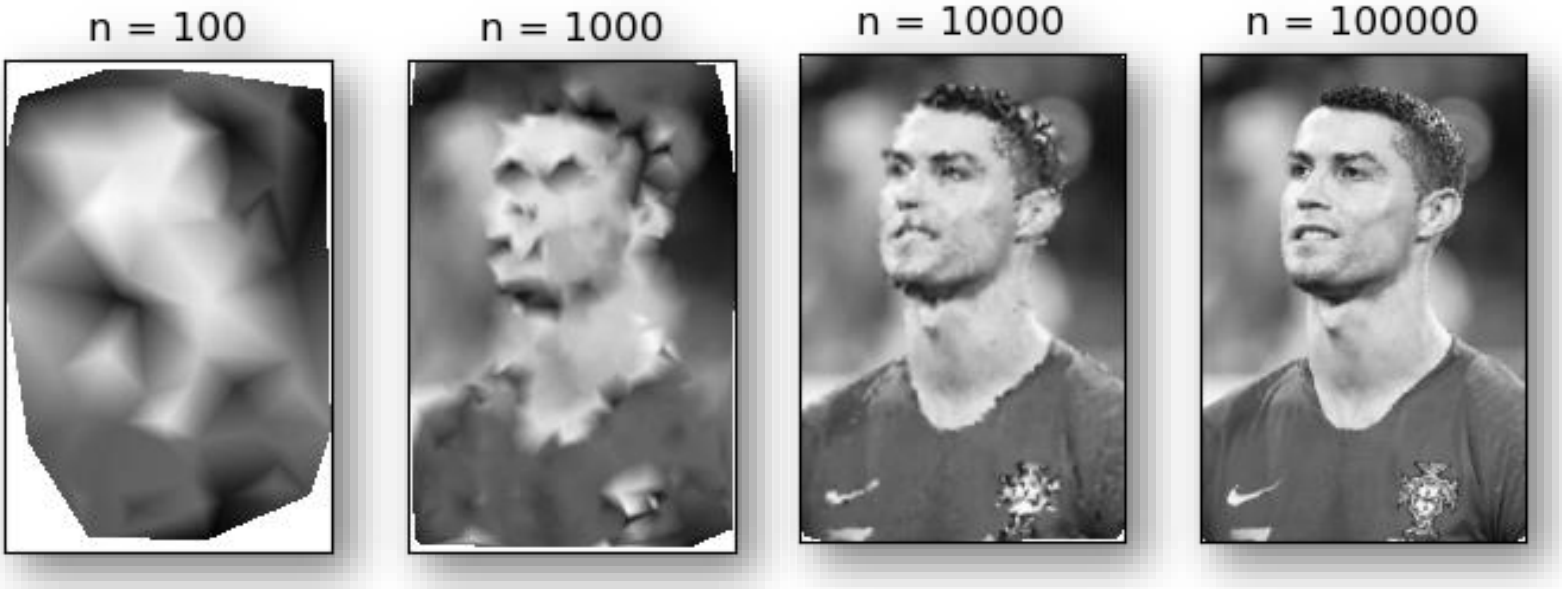


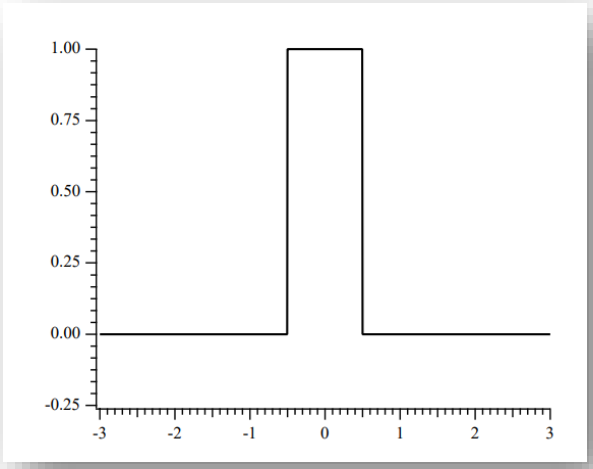
Image Interpolation Algorithms

Nearest Neighbour Interpolation Algorithm:

Because it is made up of a square pulse, the synthesis function associated with nearest-neighbour interpolation is the simplest of all.

One is the approximate order.

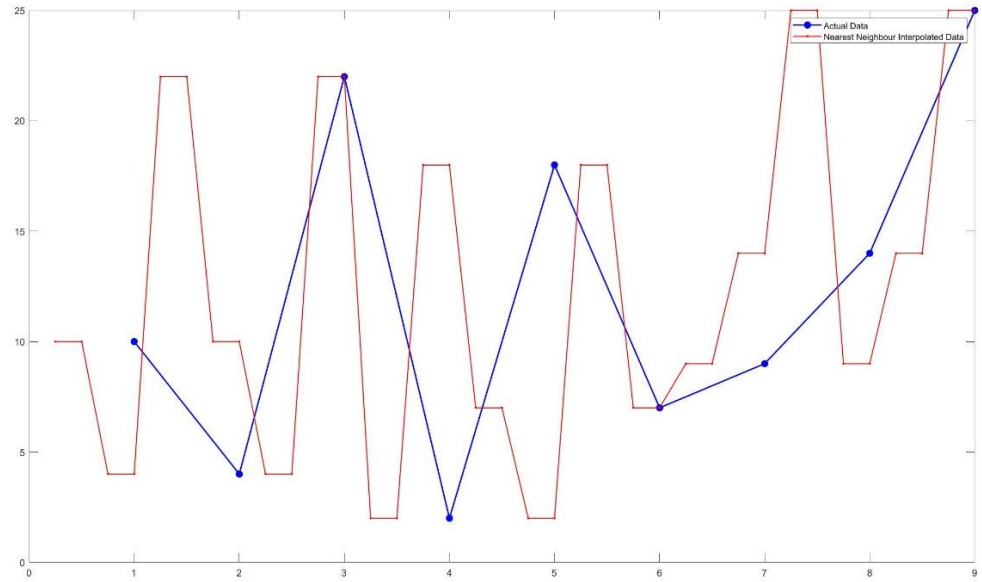
$$\varphi^0(x)=\begin{cases}0 & x<\frac{-1}{2}\\ 1 & \frac{-1}{2}\le x<\frac{1}{2}\\ 0 & \frac{1}{2}\le x\end{cases}$$



- Discontinuous
- Memory needs are minimal.
- The quickest computation time
- Each dimension requires two grid points.

MATLAB Code (.m):

```
%Nearest Neighbour Interpolation
A = zeros(3,3);
A(:)=[10,2,9,4,18,14,22,7,25];
C = imresize(A,[6,6],'nearest');
X = A';
X = X(:)';
Y = [1:1:9];
plot(Y,X,'-ob','MarkerFaceColor','b','lineWidth',1.5);
XN = C';
XN = XN(:)';
YN = [0.25:0.25:9];
hold all
plot(YN,XN,'.-r','lineWidth',1);
legend('Actual Data','Nearest Neighbour Interpolated Data')
```



Linear Interpolation Algorithm:

The linear interpolation has a rather simple implementation.

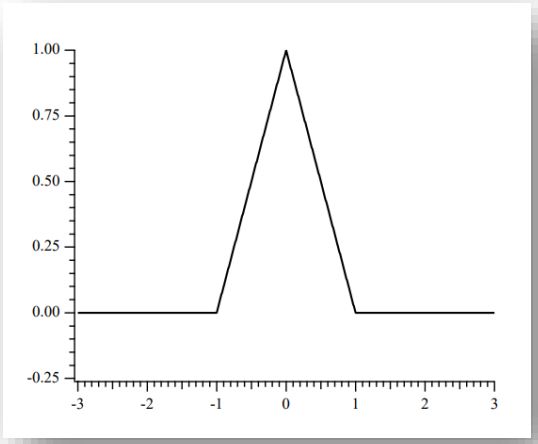
It has a two-unit support, is an interpolant, and has a two-order approximation order.

It's not differentiable, but it's continuous.

To produce an interpolated value in 1D, this interpolant takes at most two samples.

The separable implementation of 2D, also known as bilinear interpolation, necessitates four samples.

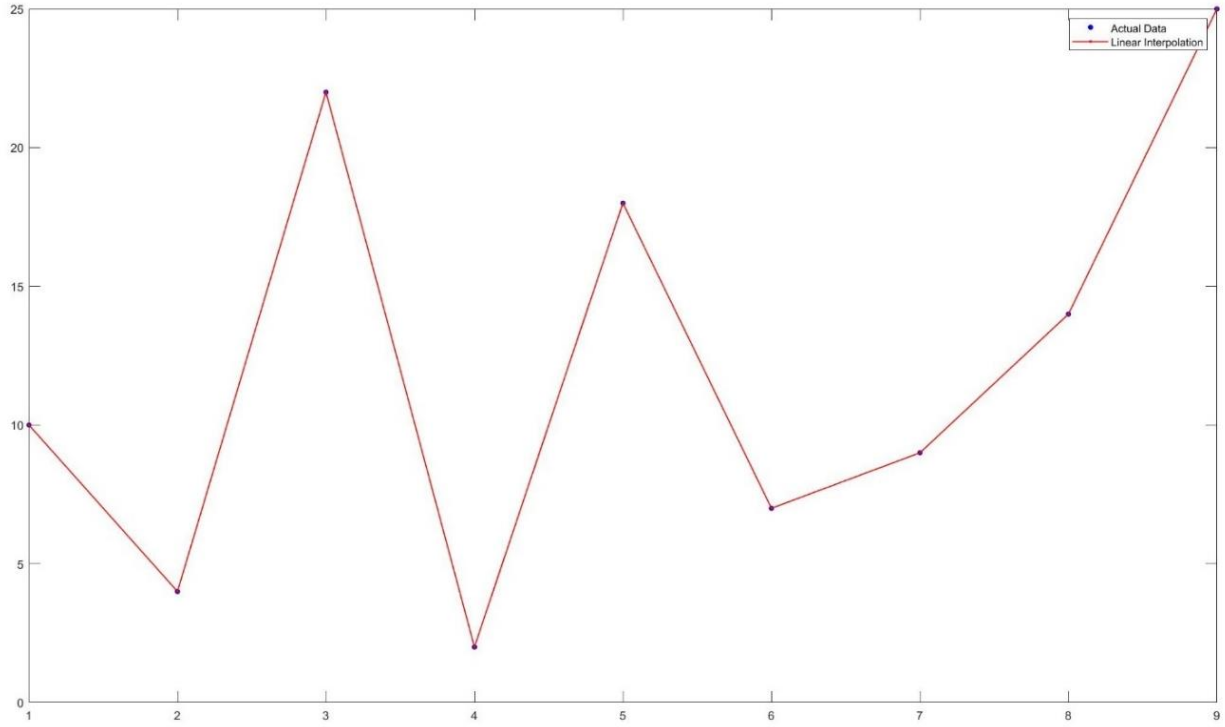
$$\beta^1(x) = \begin{cases} 1-|x| & |x|<1 \\ 0 & 1 \le x \end{cases}$$



- Continuous
- Requires at least 2 grid points in each dimension
- Requires more memory and computation time than nearest neighbour

MATLAB Code (.m):

```
%Linear Interpolation
A(:)=[10,2,9,4,18,14,22,7,25];
X = A';
X = X(:)';
Y = [1:1:9];
plot(Y,X,'.b','MarkerSize',15);
hold all
plot(Y,X,'.-r','lineWidth',1);
legend('Actual Data','Linear Interpolation')
```



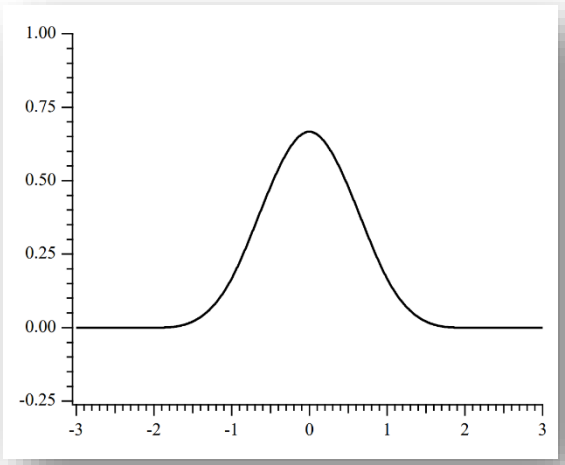
Cubic B-spline Interpolation Algorithm:

Piecewise polynomials of degree 2 are globally symmetric and 2 times continuously differentiable in this function. As a result, its regularity is C^2.

Bicubic interpolation (similar to linear interpolation) is the name given to Cubic interpolation in two dimensions.

$$\beta^3(x) = \begin{cases} \frac{2}{3} - \frac{1}{2}|x|^2\left(2 - |x|\right) & 0 \le |x| < 1 \\ \frac{1}{6}\left(2 - |x|\right)^3 & 1 \le |x| < 2 \\ 0 & 2 \le |x| \end{cases}$$

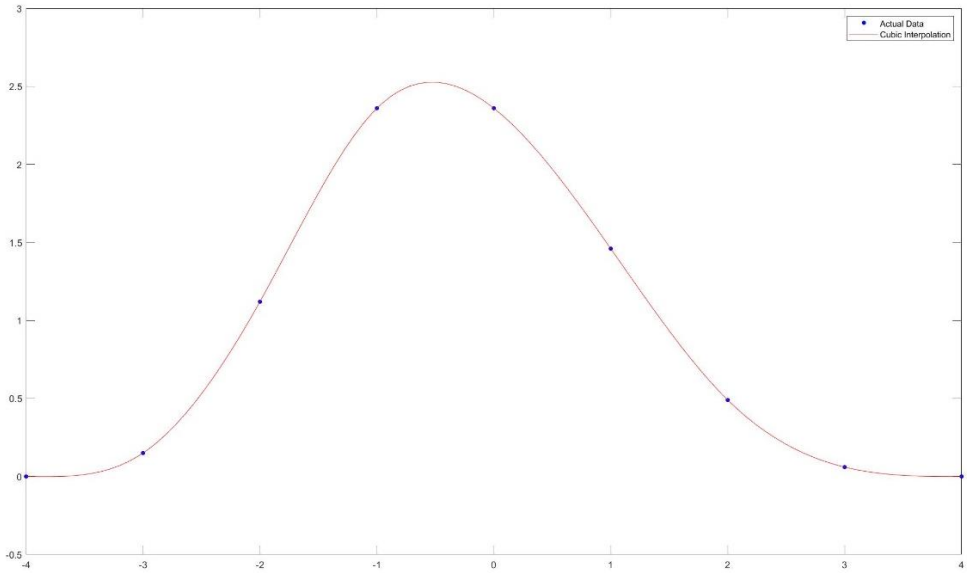
.



- Continuous
- It takes more memory and time to compute than linear.
- Although the spacing in each dimension does not have to be the same, the grid must have uniform spacing.
- Each dimension must have at least four points.

MATLAB Code (.m):

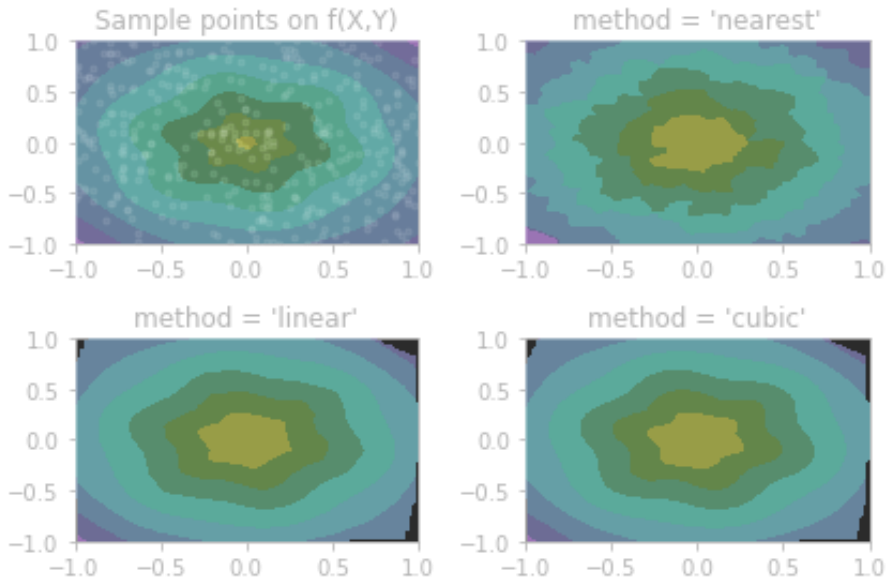
```
%Cubic Interpolation
x = -4:4;
y = [0 .15 1.12 2.36 2.36 1.46 .49 .06 0];
cs = spline(x,[0 y 0]);
xx = linspace(-4,4,101);
plot(x,y,'.b','MarkerSize',15);
hold all
plot(xx,ppval(cs,xx),'-r');
legend('Actual Data','Cubic Interpolation')
```



Comparison of different Interpolation Algorithms:

Python Code (.py):

```
import numpy as np
from scipy.interpolate import griddata
import matplotlib.pyplot as plt

x = np.linspace(-1,1,100)
y =  np.linspace(-1,1,100)
X, Y = np.meshgrid(x,y)

def f(x, y):
    s = np.hypot(x, y)
    phi = np.arctan2(y, x)
    tau = s + s*(1-s)/5 * np.sin(6*phi)
    return 5*(1-tau) + tau

T = f(X, Y)
# Choose npts random point from the discrete domain of our model function
npts = 400
px, py = np.random.choice(x, npts), np.random.choice(y, npts)

fig, ax = plt.subplots(nrows=2, ncols=2)
ax[0,0].contourf(X, Y, T)
ax[0,0].scatter(px, py, c='k', alpha=0.2, marker='.')
ax[0,0].set_title('Sample points on f(X,Y)')

# Interpolate using three different methods and plot
for i, method in enumerate(('nearest', 'linear', 'cubic')):
    Ti = griddata((px, py), f(px,py), (X, Y), method=method)
    r, c = (i+1) // 2, (i+1) % 2
    ax[r,c].contourf(X, Y, Ti)
    ax[r,c].set_title("method = '{}'".format(method))

plt.tight_layout()
plt.show()
```



Comparison of Different Interpolation Algorithms using Image Interpolation

The method of producing a continuous intensity surface from discrete image data samples is known as digital image interpolation. There are many distinct sorts of interpolation algorithms, each of which gives the finished image a different appearance.

We'll compare five algorithms in this section:

1. Nearest Neighbour Interpolation Algorithm
2. Linear Interpolation Algorithm
3. Bilinear Interpolation Algorithm
4. Bicubic Interpolation Algorithm
5. Cubic B-Spline Interpolation Algorithm

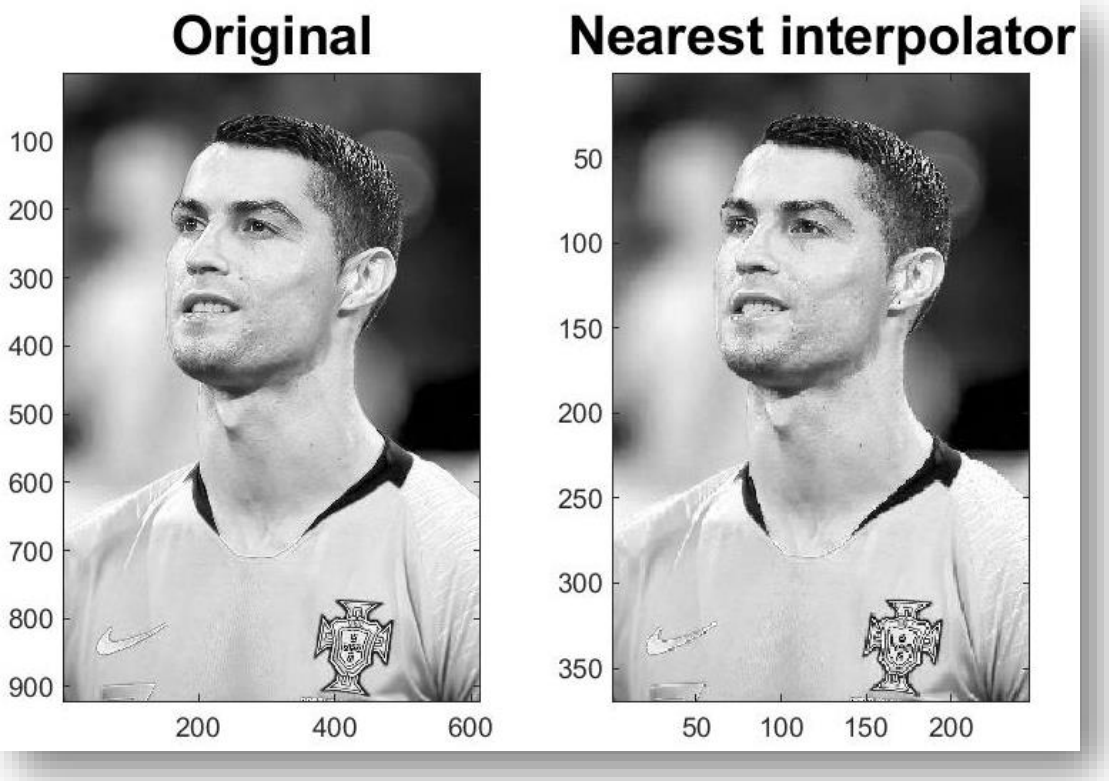
Input Image:

Cristiano Ronaldo: Wikipedia

Nearest Neighbour Interpolation Algorithm:

MATLAB Code (.m):

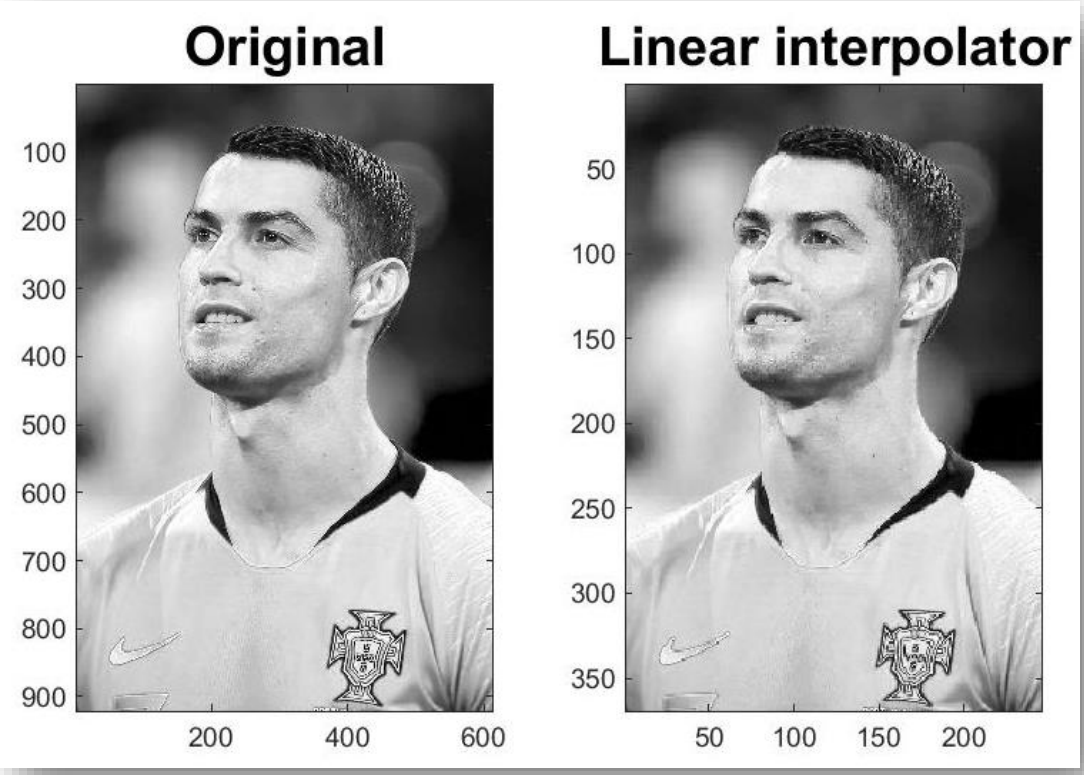
```
I=imread('CR.jpg');
I=double(I(:,:,1));

[m n]= size(I);
[x,y] = meshgrid(1:n, 1:m);

%Scaling
r=2.5;
[p,q]=meshgrid(1:r:n, 1:r:m);

%Interpolation 'linear','nearest','bilinear','bicubic','spline'
I2= interp2(x,y,I,p,q,'nearest');

figure
subplot(1,2,1),imagesc(I),axis image
title('Original','FontSize',18)
subplot(1,2,2),imagesc(I2),axis image
title('Nearest interpolator','FontSize',18)
colormap(gray)
```



Linear Interpolation Algorithm:

MATLAB Code (.m):

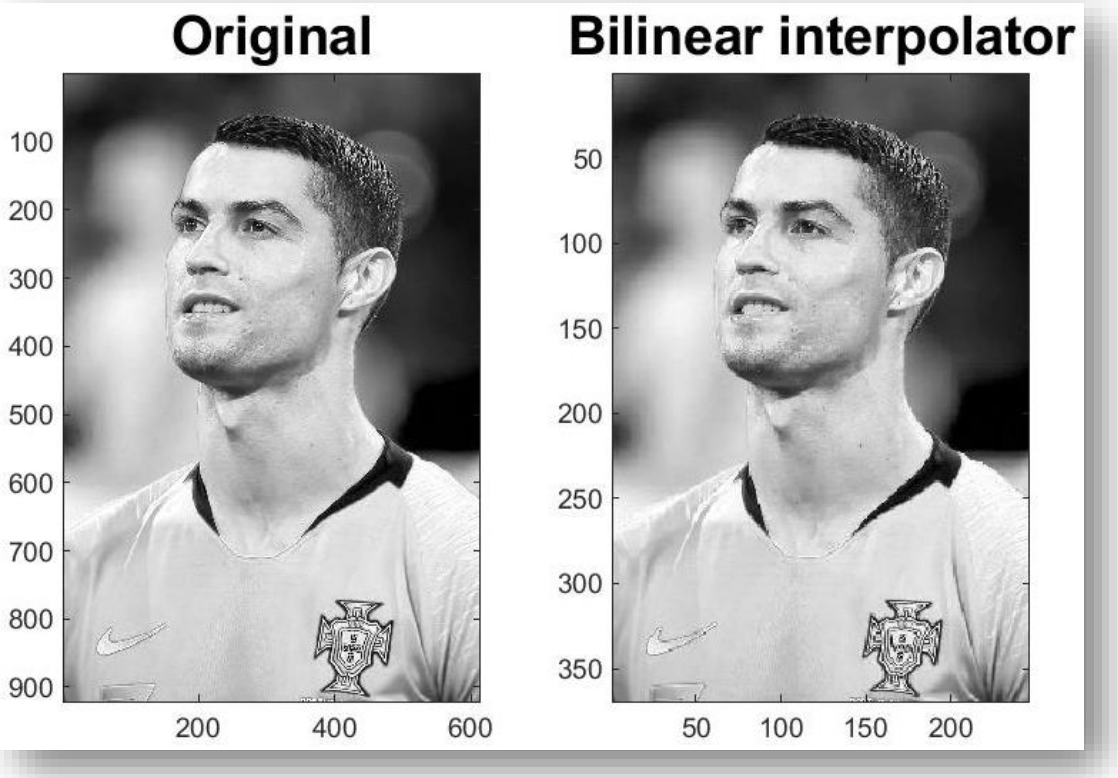
```
I=imread('CR.jpg');
I=double(I(:,:,1));

[m n]= size(I);
[x,y] = meshgrid(1:n, 1:m);

%Scaling
r=2.5;
[p,q]=meshgrid(1:r:n, 1:r:m);

%Interpolation 'linear','nearest','bilinear','bicubic','spline'
I2= interp2(x,y,I,p,q,'linear');

figure
subplot(1,2,1),imagesc(I),axis image
title('Original','FontSize',18)
subplot(1,2,2),imagesc(I2),axis image
title('Linear interpolator','FontSize',18)
colormap(gray)
```



Bilinear Interpolation Algorithm:

MATLAB Code (.m):

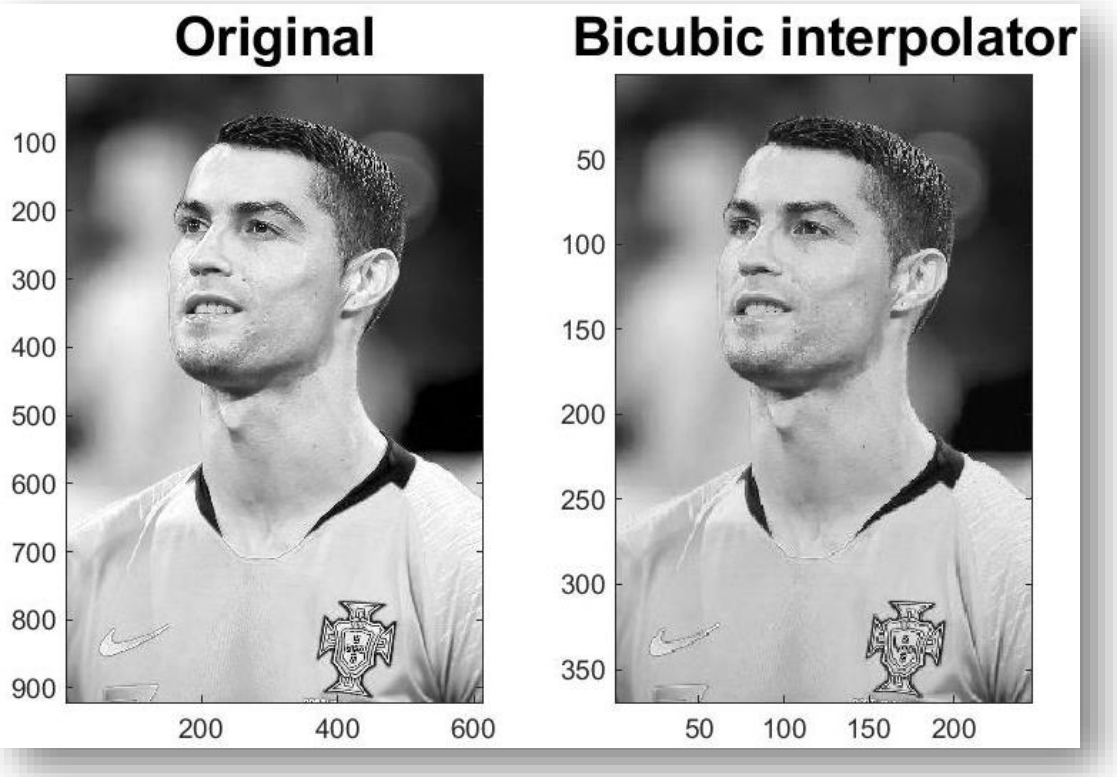
```
I=imread('CR.jpg');
I=double(I(:,:,1));

[m n]= size(I);
[x,y] = meshgrid(1:n, 1:m);

%Scaling
r=2.5;
[p,q]=meshgrid(1:r:n, 1:r:m);

%Interpolation 'linear','nearest','bilinear','bicubic','spline'
I2= interp2(x,y,I,p,q,'bilinear');

figure
subplot(1,2,1),imagesc(I),axis image
title('Original','FontSize',18)
subplot(1,2,2),imagesc(I2),axis image
title('Bilinear interpolator','FontSize',18)
colormap(gray)
```



Bicubic Interpolation Algorithm:

MATLAB Code (.m):

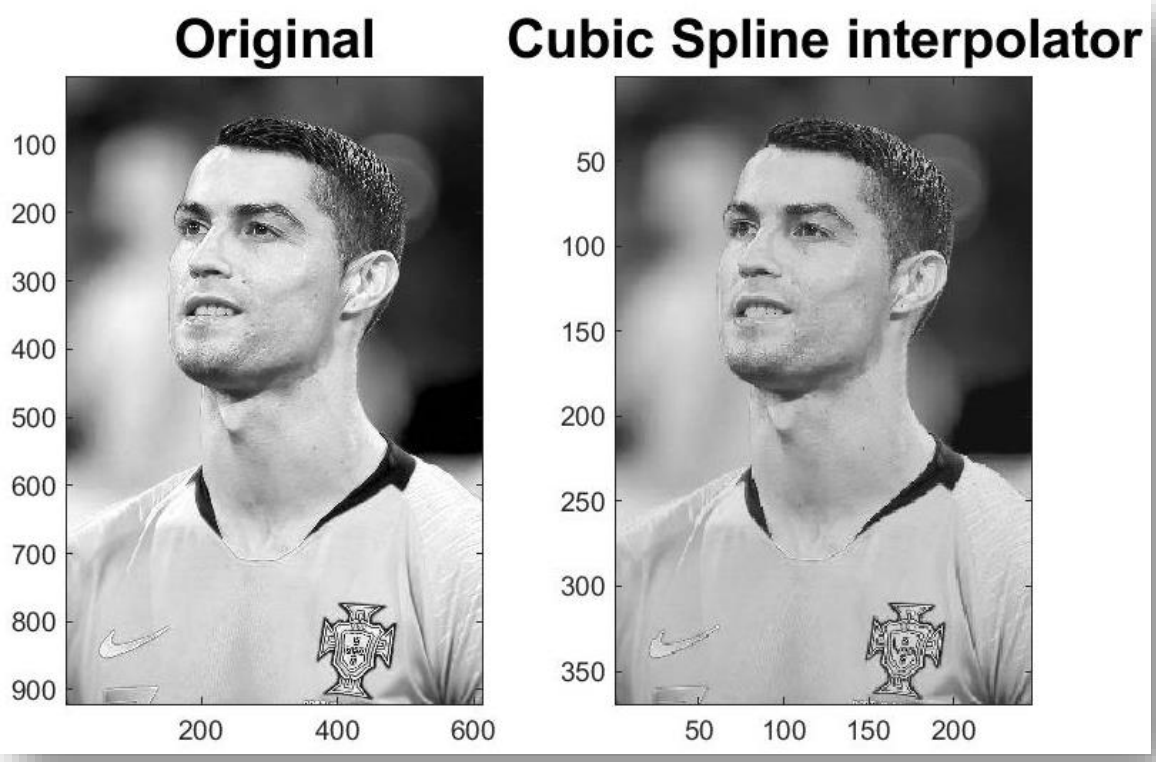
```
I=imread('CR.jpg');
I=double(I(:,:,1));

[m n]= size(I);
[x,y] = meshgrid(1:n, 1:m);

%Scaling
r=2.5;
[p,q]=meshgrid(1:r:n, 1:r:m);

%Interpolation 'linear','nearest','bilinear','bicubic','spline'
I2= interp2(x,y,I,p,q,'bicubic');

figure
subplot(1,2,1),imagesc(I),axis image
title('Original','FontSize',18)
subplot(1,2,2),imagesc(I2),axis image
title('Bicubic interpolator','FontSize',18)
colormap(gray)
```



Cubic B-Spline Interpolation Algorithm:

MATLAB Code (.m):

```
I=imread('CR.jpg');
I=double(I(:,:,1));

[m n]= size(I);
[x,y] = meshgrid(1:n, 1:m);

%Scaling
r=2.5;
[p,q]=meshgrid(1:r:n, 1:r:m);

%Interpolation 'linear','nearest','bilinear','bicubic','spline'
I2= interp2(x,y,I,p,q,'spline');

figure
subplot(1,2,1),imagesc(I),axis image
title('Original','FontSize',18)
subplot(1,2,2),imagesc(I2),axis image
title('Cubic Spline interpolator','FontSize',18)
colormap(gray)
```



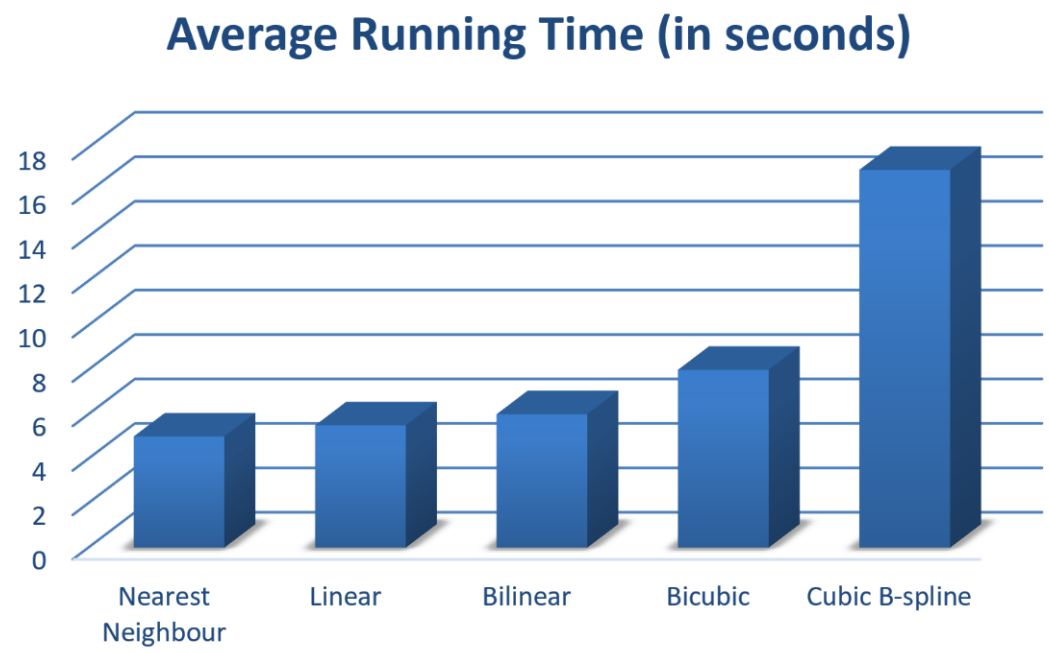
Conclusions

Time Complexity of different Algorithms are given as:

Interpolation Algorithms	**Time in Seconds (s)**
Nearest Neighbour	0.081
Linear	0.099
Bilinear	0.140
Bicubic	0.169
Cubic B-spline	0.174

Evaluation of Different Interpolation Algorithms:

Interpolation Algorithm	**Subjective Feelings**	**Evaluation (Reviews)**	**Processing Time (seconds)**
Nearest Neighbour	Obvious mosaic	Worst	5
Linear	Very Blurry	Worse	5.5
Bilinear	Blur, not Sharp	Bad	6
Bicubic	Fuzzy, Sharper	Better	8
Cubic B-spline	Relatively Clear, Sharp	Good	17



Nearest Neighbour Interpolation Algorithm:

- Most Simple and Fastest
- Brings Significant Distortion

Linear and Bilinear Interpolation Algorithm:

- More Complex than Nearest Neighbour
- Low Pass filtering properties
- No gray discontinuity defects

Bicubic Interpolation Algorithm:

- Clear output picture quality
- Larger amount of calculations
- Most commonly used in Photoshop, After Effect, etc.

Cubic B-Spline Interpolation Algorithm:

- Smooth image outcome
- No obvious saw tooth phenomenon
- When amplification factor is higher, it causes fuzzy edge and false traces.

References

- wikipedia.org
- math.stackexchange.com
- in.mathworks.com
- scipy.org
- RK Jain and SRK Iyengar, *"Numerical Methods"*
- Dianyuan Han, *"Comparison of Commonly Used Image Interpolation Method",* 2nd ICCSEE 2013
- Mr. Pankaj S. Parsania, Dr. Paresh V. Virparia, *"A Comparative Analysis of Image Interpolation Algorithms",* IJARCCE Vol. 5, Issue 1, January 2016
- Philippe Thévenaz, Thierry Blu and Michael Unser, *"Image Interpolation and Resampling",* Swiss Federal Institute of Technology—Lausanne