# L5 (Image Interpolation)

The goal of image interpolation is to used already available data to estimate sample values at new pixels. Image interpolation might be necessary when carrying out the following operations: resizing, up-sampling, rotation, scaling, affine transformations, homographies
The same for video: interpolate samples in temporal direction
**Equally sampled points**:

- points are at regular intervals
- key advantage: allows separability, meaning we can process rows first, then columns (or vice versa)
- Nearest neighbor, bilinear, bicubic, Lanczos, or splines

**Non-equally sampled points**:

- points are at irregular intervals: no assumption can be made about sample locations
- relies on triangulation techniques to do more generic interpolation

## Image Upsampling

**Goal**: increase image resolution by a factor of $L$ when both input and output points are equally spaced.
Three steps:

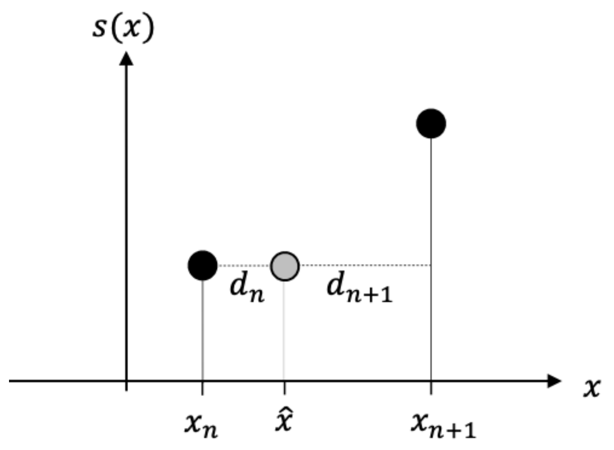
1. Zero-insertion: to create space for new pixels
2. Row interpolation: apply filter $h_r[n_1]$ to interpolate along rows
3. Column interpolation: apply filter $h_c[n_2]$ to interpolate along columns

When we insert zeros, we introduce high-frequency artifacts in the signal. Convolution $h[m,n]$ acts as a low-pass filter to smooth out the discontinuities created by zero insertion. It fills in the zero values meaningful pixel values based on neighboring pixels.
The filter needs a gain of $L^2$ to compensate for energy loss. This is because inserting zeros reduces the signal energy, and we need to compensate for this loss in both dimensions.
$h[m,n] = h_r[m]h_c[n]$ allows for efficient computation by applying row and column filters separately rather than a single 2D convolution

## Nearest Neighbor Interpolation

- copy the value of the nearest available neighbor sample

- produces distortions of straight edges



$$s(\hat{x}) = \begin{cases} s(x_n) & d_n \le d_{n+1} \\ s(x_{n+1}) & d_{n+1} < d_n \end{cases}$$

### Nearest Neighbor Image Upsampling

**Zero-order hold** filter

- repeat pixels along scan line: $h[m,n] = 1_{LxL}$

  Example ($L$=2): $h[m,n] = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 5 & 6 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 3 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 5 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 1 & 3 & 3 & 2 & 2 \\ 1 & 1 & 3 & 3 & 2 & 2 \\ 4 & 4 & 5 & 5 & 6 & 6 \\ 4 & 4 & 5 & 5 & 6 & 6 \end{bmatrix}$$

## Bilinear Interpolation

It fits a straight line between two consecutive samples in 1D:

$$s(\hat{x}) = d_{n+1}s(x_n) + d_n s(x_{n+1})$$

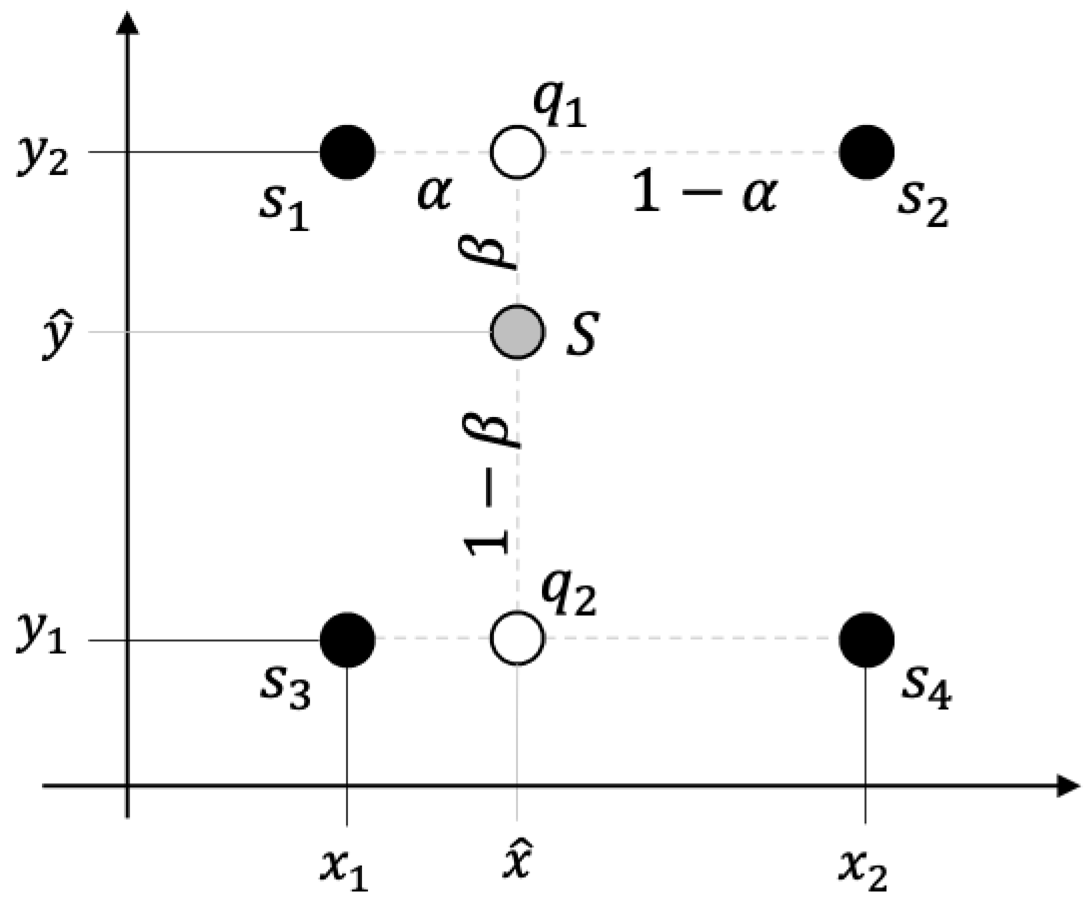
For image signals in 2D:

$$q_1 = (1-\alpha)s_1 + \alpha s_2$$

$$q_2 = (1-\alpha)s_3 + \alpha s_4$$

Bilinear interpolation: first applied to rows then to columns (separability)

$$s = (1-\beta)q_1 + \beta q_2 = (1-\beta)(1-\alpha)s_1 + (1-\beta)\alpha s_2 + \beta(1-\alpha)s_3 + \beta\alpha s_4$$



We can interpret it as an area-based weighted linear combination

## Linear Interpolation

Fits straight line between pixels along row/column:

$$h_r[m] = h_c^T[n] = \frac{1}{L}[1, 2, \ldots, L-1, L, L-1, \ldots, 2, 1]$$

If $L = 2$, then:
$h_r[m] = h_c^T[n] = [\frac{1}{2}, 1, \frac{1}{2}]$
The separable filter matrix is computed by matrix product of $h_r[m]$ and $h_c^T[n]$:

$$h[m, n] = h_r[m]h_c^T[n]$$

The next step is to do zero insertion to the original image to achieve $(L * m) \times (L * n)$ matrix $(3 \times 3 \to 9 \times 9)$
Finally, convolve this new upsampled image with $h[m, n]$ filter placing the filter center at the point of interest. Consider zero padding if the point is on the border(standard practice)

## Ideal Interpolation

It is achieved via sinc filter which maintains frequency content. It doesn't affect low frequencies, doesn't add high frequencies.

L5

$$h(x) = \frac{\sin(\pi x)}{\pi x} = sinc(x)$$

In the context of digital signal processing:

$$h[n] = sinc[n/L]$$

where $L$ is the upsampling factor.
The Fourier transform of a sinc function is a perfect rectangular function (box function) and this allows it to act as ideal low-pass filter.
**Problems:**

- infinite summation required
- blurring due to finite signal length and instationarity of image signals

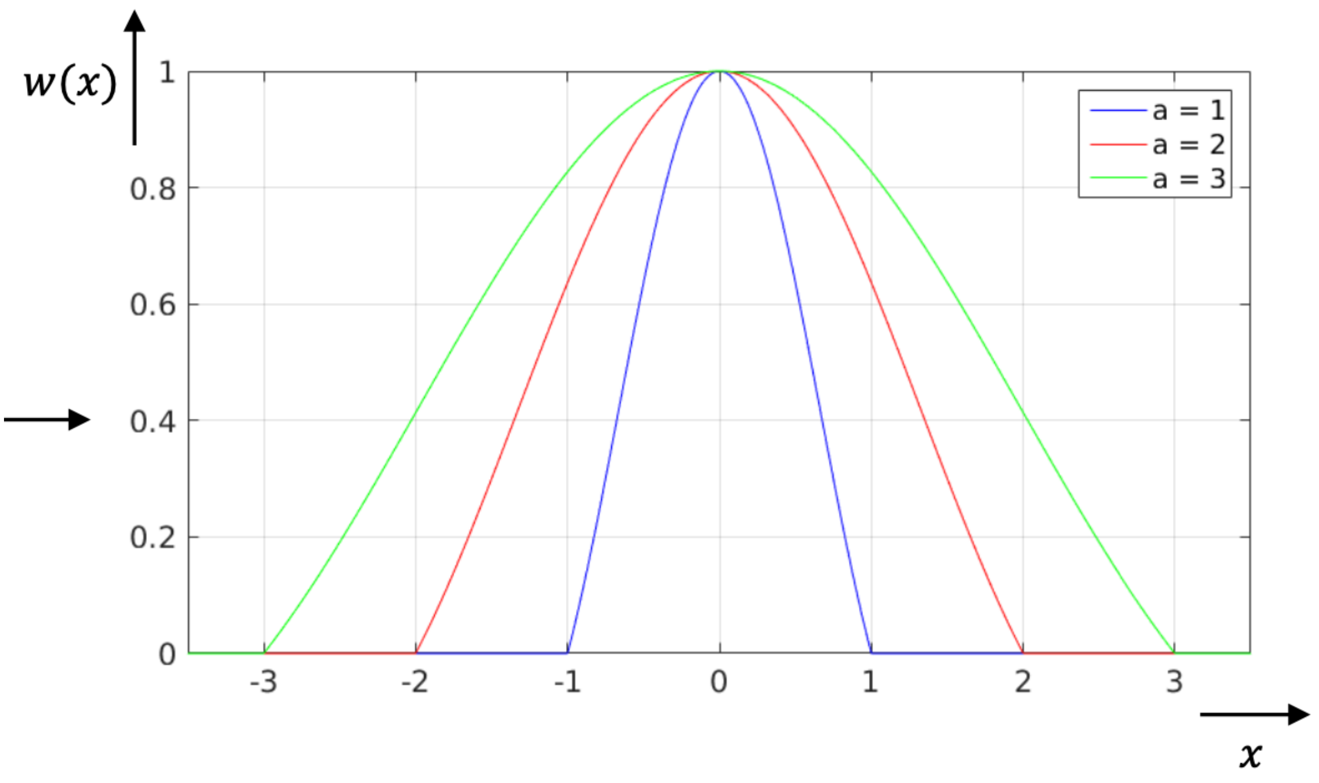
## Lanczos Interpolator

It is an approximation of **sinc** filter. It does so by **Lanczos window**:

$$w(x) = sinc(\frac{x}{a}) \quad \text{for} - a < x < a$$

$$w(x) = 0 \quad \text{otherwise}$$

$a$ determines filter width:



**Lanczos kernel**: complete interpolation function:

$$sinc(x) \quad sinc(\frac{x}{a})$$

So, Lanczos kernel is a multiplication of sinc function and Lanczos window.

Increasing $a$ provides more flexibility for shaping the frequency response, but it also increases computational cost. Lanczos kernel is **non-separable.**

## Spline Interpolation

A method that fits piecewise polynomial functions between data points to create smooth curves.
It uses polynomial pieces between each pair of data points and these pieces are joined smoothly at "knots" (the original data points). This ensures the continuity of the function and its derivatives at the knots.
Key advantage of splines over high-order polynomials is that it minimizes oscillations. There are multiple types of splines: linear, quadratic, cubic.

### B-splines

B-splines of higher orders are constructed through recursion:

$$\phi_n(x) = \phi_0(x) * \phi_{n-1}(x)$$

where $\phi_0(x)$ is the zero-order B-spline (box function)
Then:

- $\phi_1(x) = \phi_0(x) * \phi_0(x)$: convolution of box with itself
- each convolution makes the function smoother
- peak values decrease( 1->1->0.75->2/3) and the x-support increase: 1/2->1->1.5->2.0
- the function becomes more bell-shaped

### Linear Splines

Each point between two samples is influenced by TWO overlapping splines:

$$s(x) = c_n \phi_1(x - x_n) + c_{n+1} \phi_1(x - x_{n+1})$$

where $c_n$ and $c_{n+1}$ are coefficient for each spline. $\phi_1$ represents the linear B-spline basis function. $x - x_n$ shifts the spline to the correct position

### Cubic Splines

Each point is now influenced by FOUR overlapping splines. It provides smoother interpolation than linear, as there is contribution from four consecutive points:

$$s(x) = c_{n-1}\phi_3(x - x_{n-1}) + c_n \phi_3(x - x_n) + c_{n+1}\phi_3(x - x_{n+1}) + c_{n+2}\phi_3(x - x_{n+2})$$

Each $x_i$ is accompanied by its coefficient $c_i$. Notice the index starts from $n-1$ and ends at $n+2$.

### Constraint:

To ensure that the spline interpolation **exactly** matches the given discrete data samples $g(x_n)$, we impose the interpolation condition:

$$s(x_n) = \frac{1}{6}(c_{n-1} + 4c_n + c_{n+1})$$

$c_{n+2}$ is omitted because it is outside the support of cubic spline. Its contribution becomes zero.

## Problem solving

- For an $L$ spline interpolation, we need $L+1$ basis splines
- Each basis spline is calculated based on recursive definition above
  The interpolated value is regulated by 4 points for a cubic spline. So $s(x)$ must be calculated by the closest 4 whole digit numbers to $x$. For example, $g(1.4)$ would be:

$$g(1.4) = \sum_{i}^{3} c_i^3 * \phi(1.4 - x_i)$$

# Triangulation

The setup here is that samples are located at **arbitrary** locations which means there is no regularity and no separability. It's not clear which 4 points to consider for bilinear interpolation. The solution is **triangulation**.

- each unknown pixel is enclosed by a triangle
- many possible triangulations exist, which one to choose?

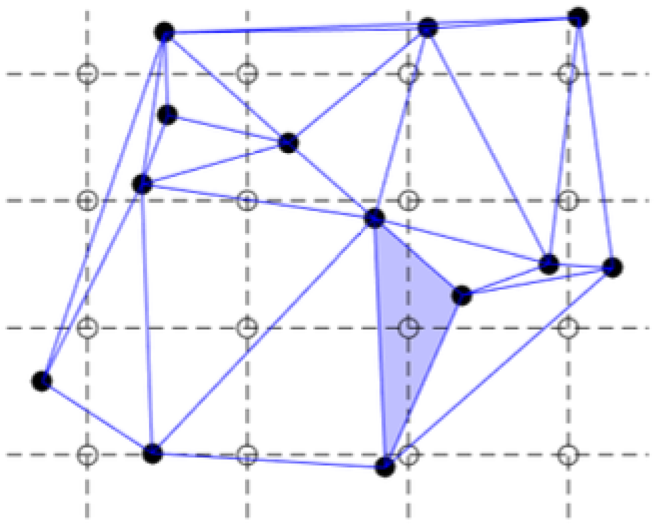
## Delauney triangulation

- Maximizes the minimum angles in a triangle (avoids extremely thin triangles)
- Circumcircle of a triangle contains only its vertices (and no other sample point vertices, only 3, not 4!)

Linear interpolation with Delauney triangulation: each triangle defines a plane in 3D using the equation $ax + by + cz + d = 0$. Three points of each triangle define a linear surface. The

$z-$ coordinate represents the pixel value or color at that point.
To find an interpolated value at any position within the triangulated area:

- determine which triangle contains the query point
- evaluate the plane equation at that position to get the interpolated $z-$ value



## Worked Example

Suppose we have three known points forming a triangle:

- $P_1 = [1, 1, 50]$
- $P_2 = [3, 1, 100]$
- $P_3 = [2, 3, 25]$
  We want to find the **z-value** of at point $Q = [2, 2, z]$
- First, we solve the plane equation $ax + by + cz + d = 0$ using the given three points.
- Then, we calculate **z-value** using the computed coefficients.

## Voronoi Tessellation

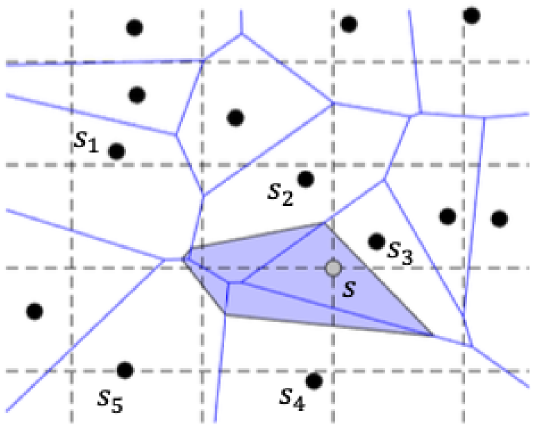
A method of dividing space into regions based on proximity to points. Each Voronoi cell contains all locations that are closer to its sample point $p$ than to any other sample point. Voronoi diagram is the dual graph of the Delauney Triangulation. It's constructed by connecting the centers of circumcircles of each Delauney triangle.

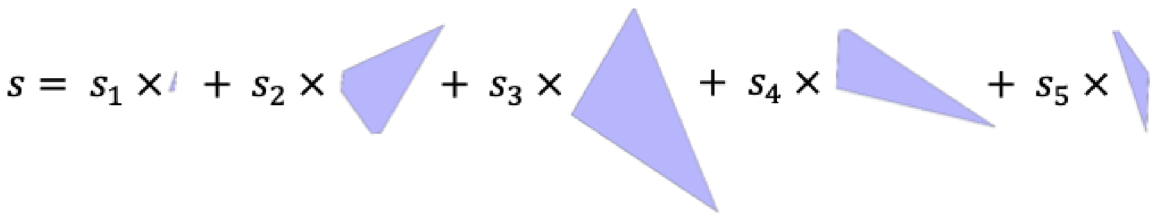
## Natural Neighbor Interpolation

Interpolation based on Voronoi Tessellation is more natural than linear interpolation because:

- it considers neighboring points

- it produces smoother transitions between values
- it preserves local features of the data better.




$s = s_1 \times + s_2 \times + s_3 \times + s_4 \times + s_5 \times$

Process:

- We have our existing Voronoi diagram (in blue lines)
- Then, we temporarily insert query point for interpolation. This creates a new Voronoi cell $s$.
- This new cell "steals" the area from the neighboring cells.
  So, the interpolated value is then a weighted combination of neighboring cell values where the weights are controlled by the proportions of area stolen from each neighbor and they are normalized to $1$.