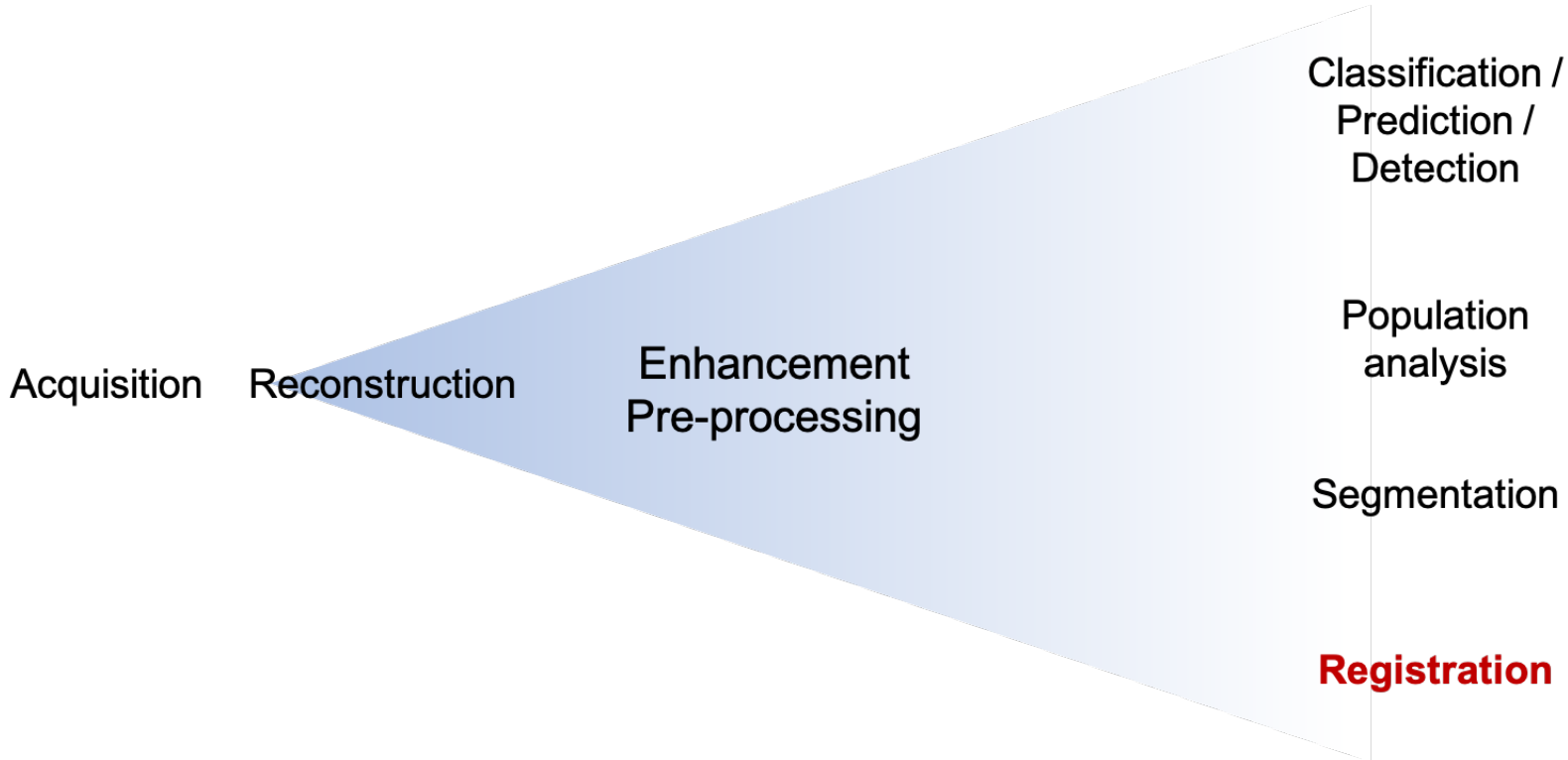
Spatial Transformations

Ender Konukoglu

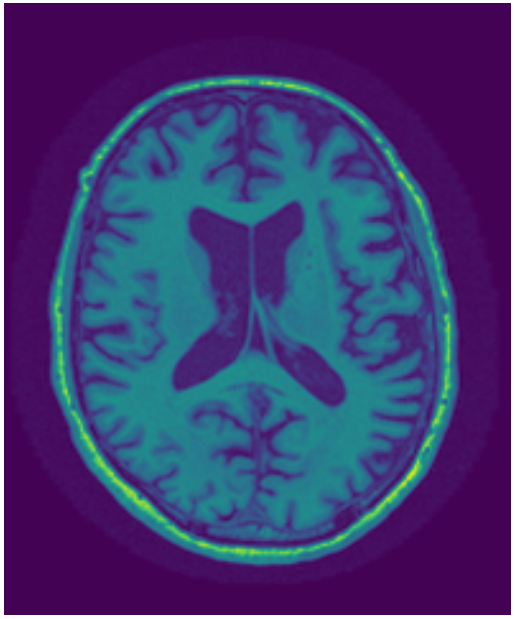
ETH Zürich

March 03, 2020

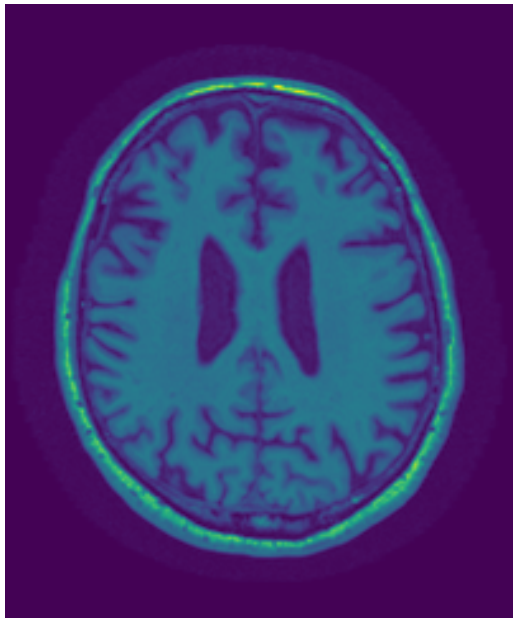
Registration is an essential task in medical image analysis



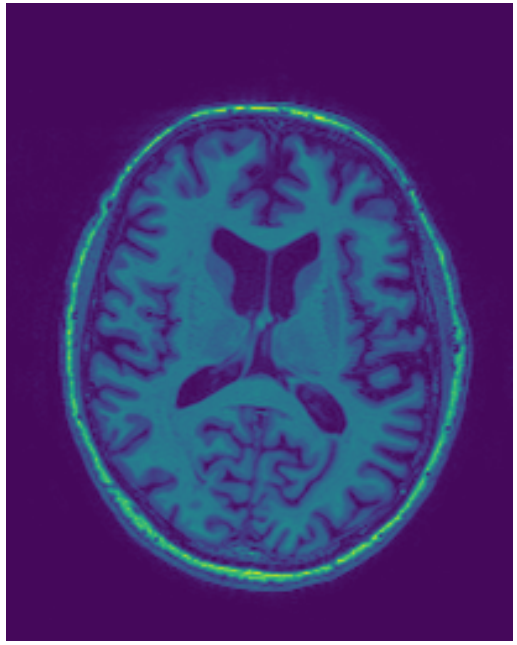
What is image registration and why do we need it?



Time point 1



Time point 2

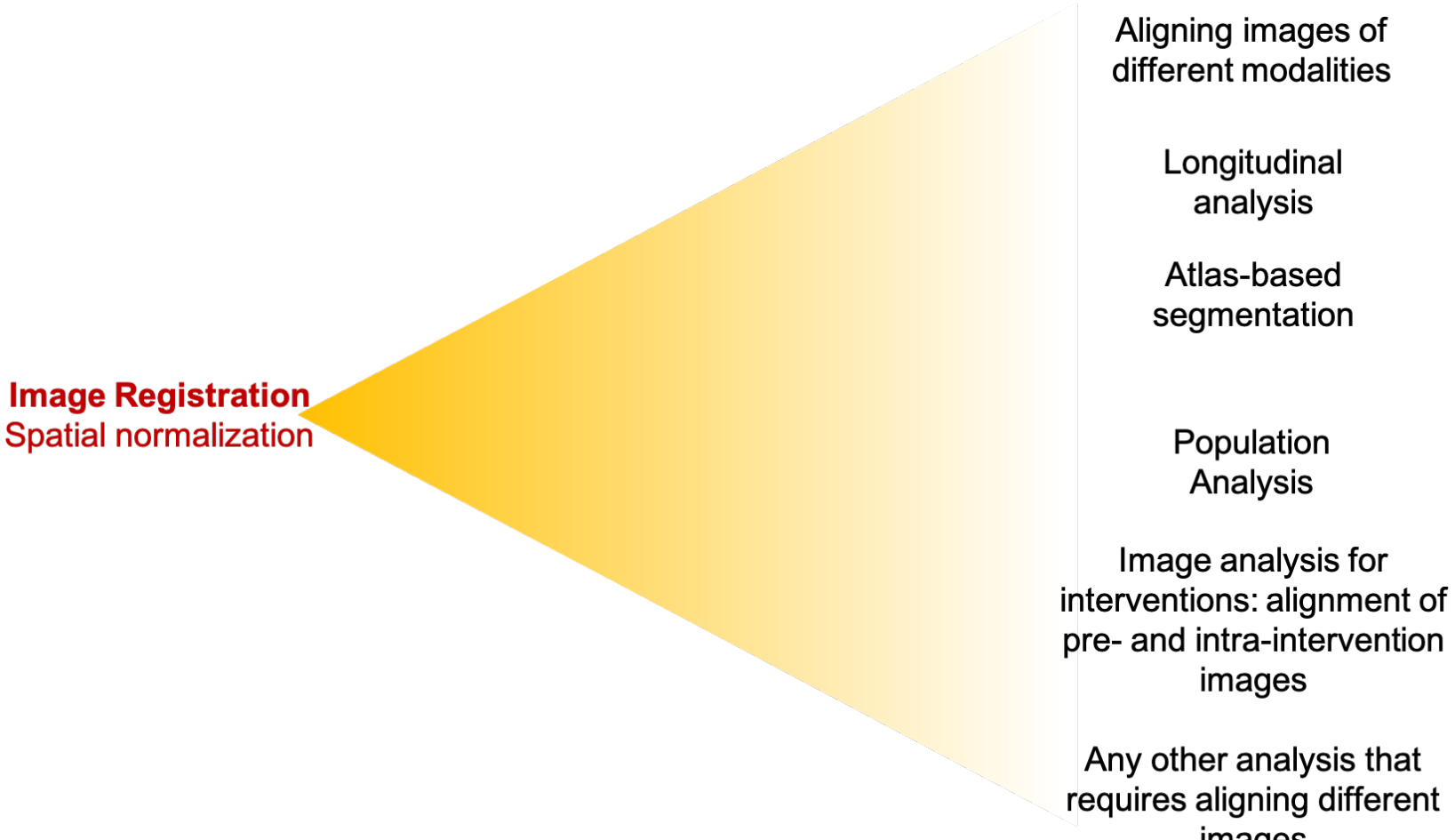


Time point 3

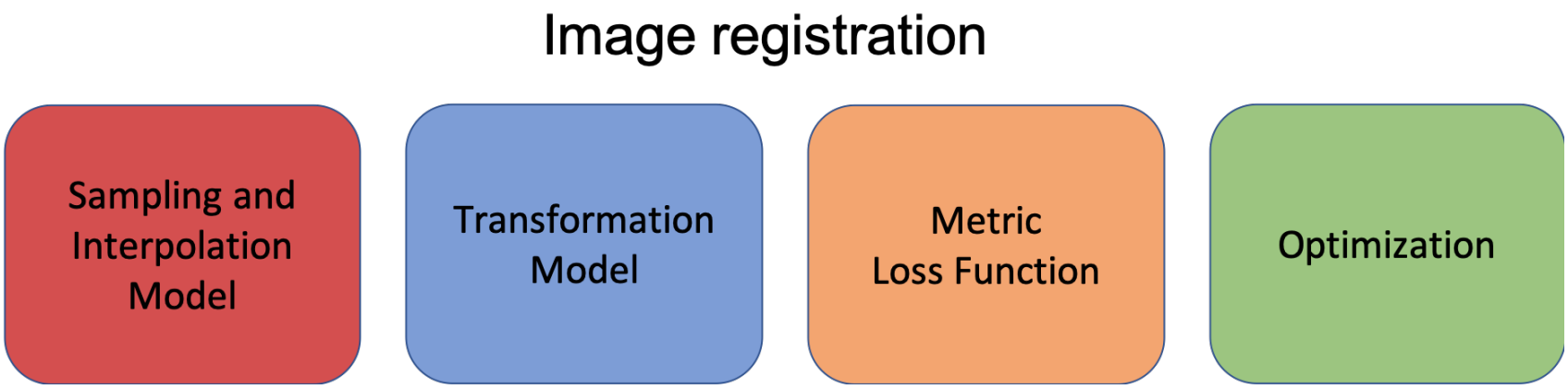
A common problem

These are three images taken from the same individual 6 months apart. Images show the same cross-section (90th slide) from three different volumetric images. Notice that they do not show the same anatomy. That is because they are not aligned. Image registration aligns these images through spatial transformations and allows comparisons.

Useful for many different applications



Four main components



Any registration algorithm is composed of four components. Today we will study the "Sampling and interpolation model" and start "Transformation model".

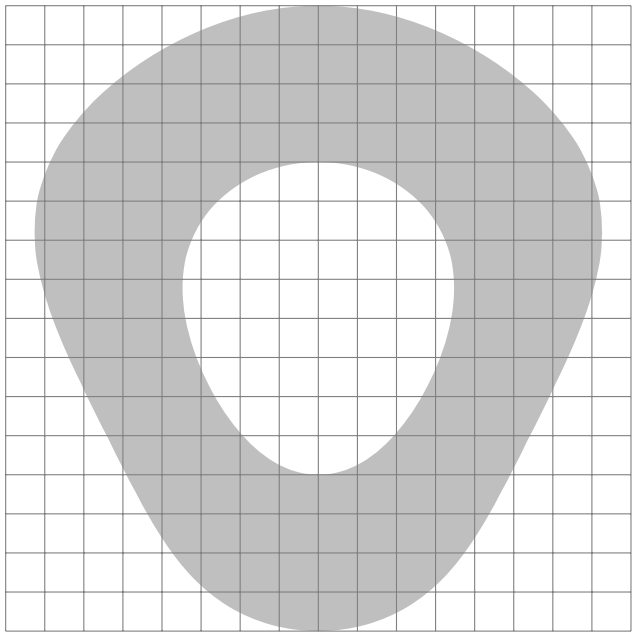
Outline

- Sampling model - How to apply a transformation
- Interpolation models
- Linear transformation models
- Non-linear transformation models

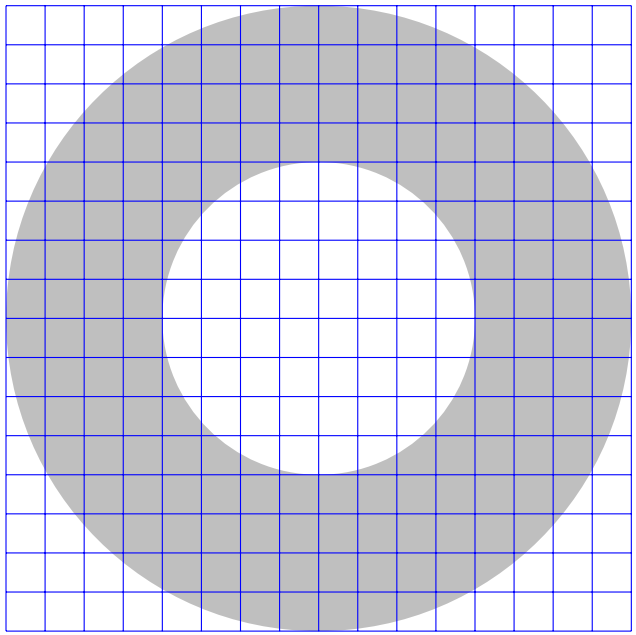
Section 2

Sampling model

Setup



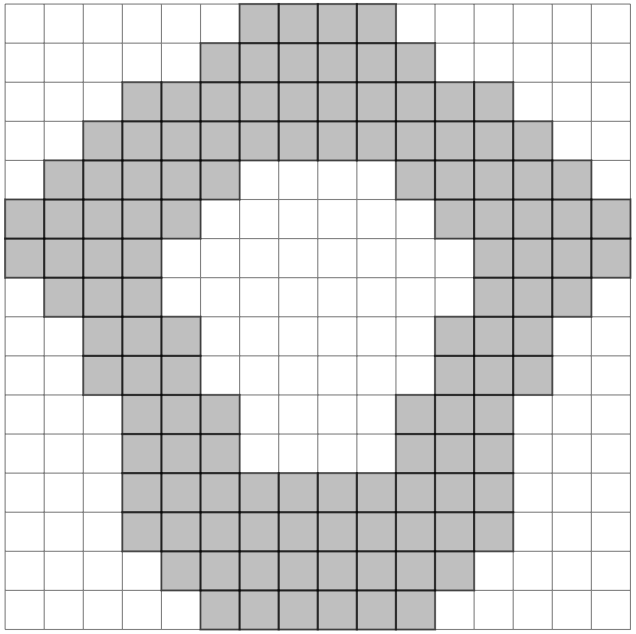
Moving Image / Input Image



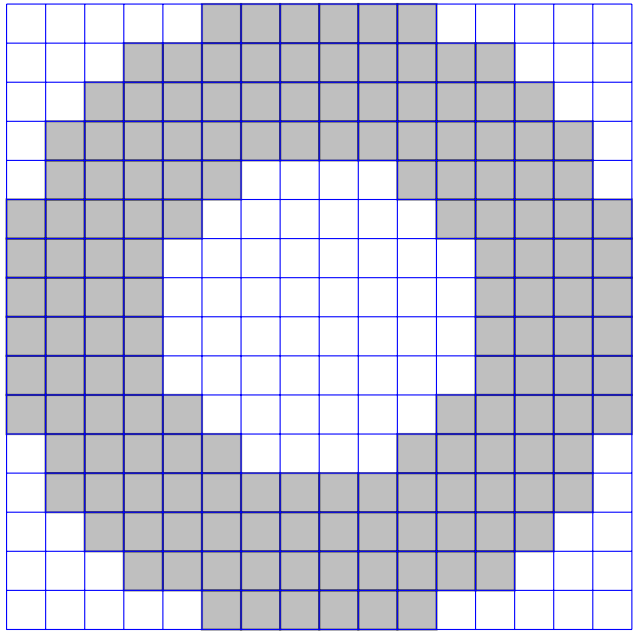
Target Image

- Moving image is the one that will be transformed.
- Target image is to target that we want to attain with the transformation.
- Both images have their own pixel coordinates and corresponding Cartesian grids.
- Assume there exists a spatial transformation that matches the moving to the target image.

Setup: Discrete



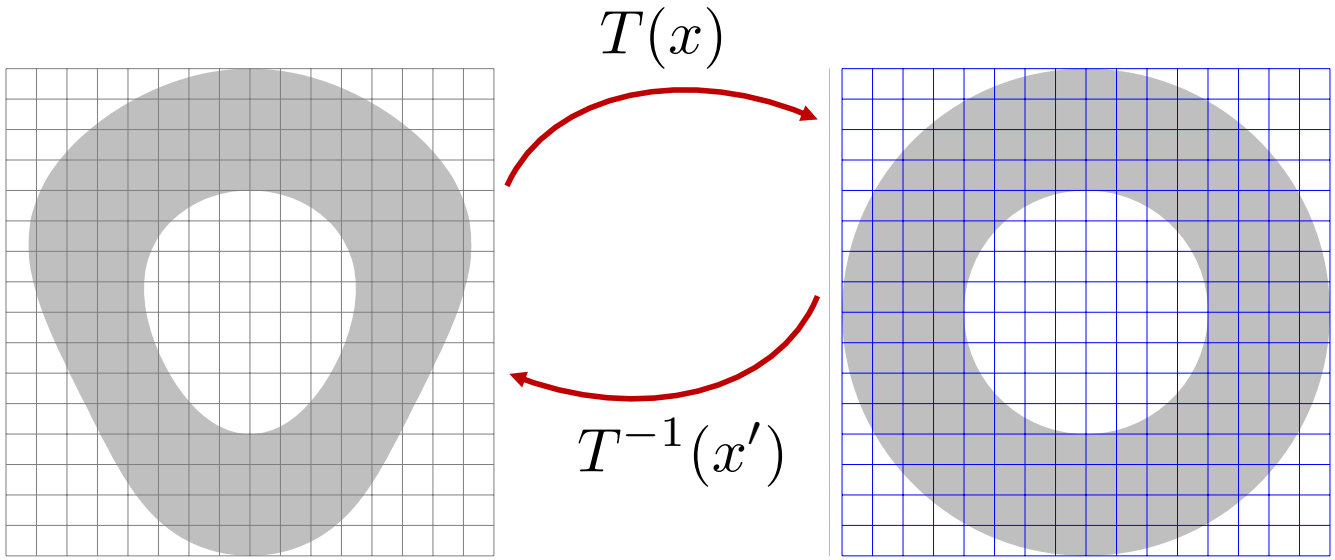
Moving Image / Input Image



Target Image

- Moving image is the one that will be transformed.
- Target image is to target that we want to attain with the transformation.
- Both images have their own pixel coordinates and corresponding Cartesian grids.
- Assume there exists a spatial transformation that matches the moving to the target image.
- In reality images are discretized - we only have values at pixel locations.

Transformation

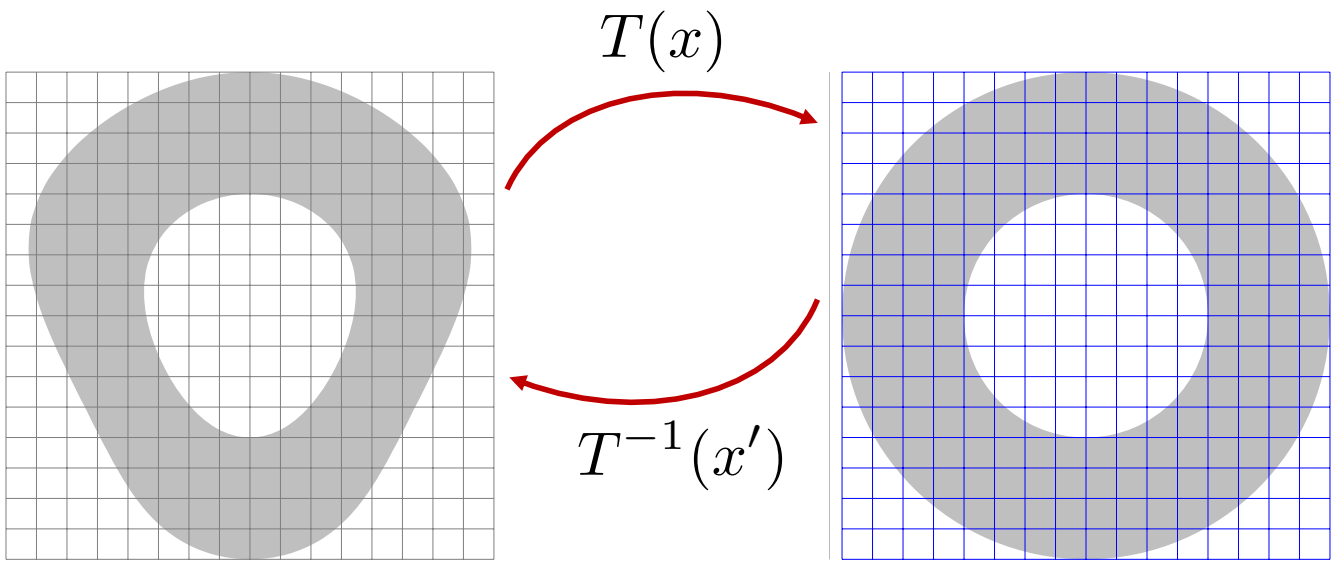


Let us assume $x' = T(x)$ is the transformation model with $x, x' \in \mathbb{R}^2$ in this case, where x and x' represents points in the moving and target image coordinate systems, respectively. Discussions for 3D transformations is the same.

Let us also define $x = T^{-1}(x')$ as the inverse of the transformation model.

Number of possible models: $T(x) = Ax$ (linear) or $T(x) = x + u(x)$ (non-linear)

Main question for the sampling model



Main sampling model question

Given the images, respective pixel-grids and transformations, how do we generate the "output" transformed moving image?

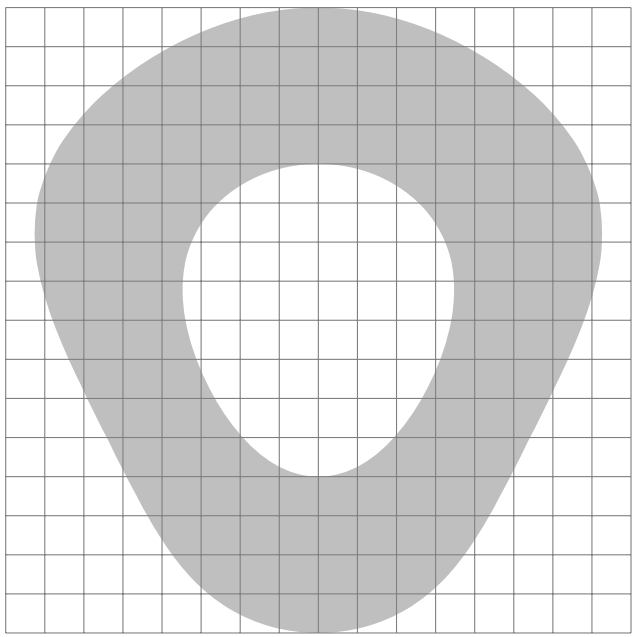
There are two possibilities:

- Forward transformation
- Backward transformation

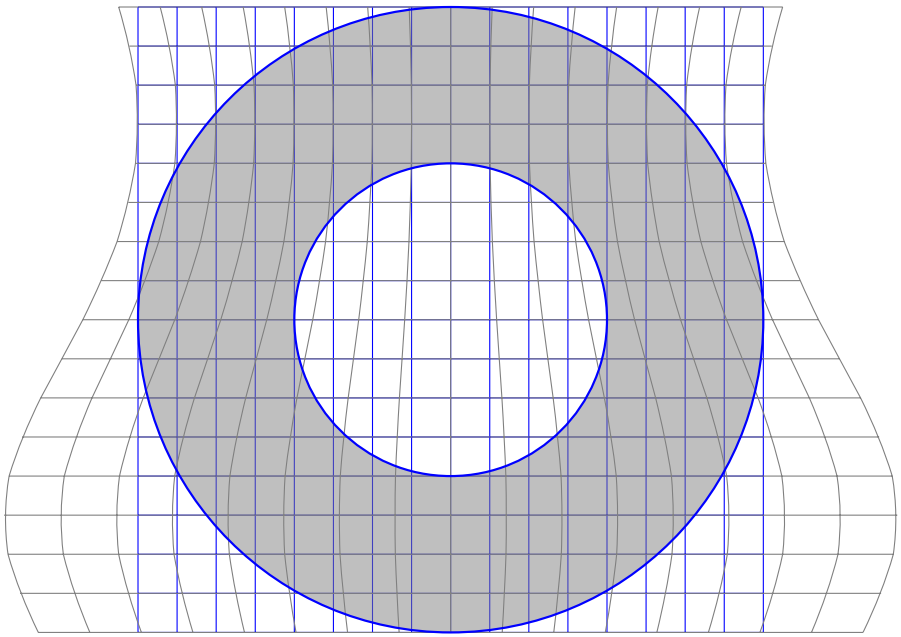
Forward transformation

Let us use the following forward transformation / mapping

$$T(x) = \{x_1 / [0.15 \sin(2.5 x_2) + 0.85], x_2\} = \{x_1', x_2'\} = x'$$



Input grid



Transformed grid and Output grid

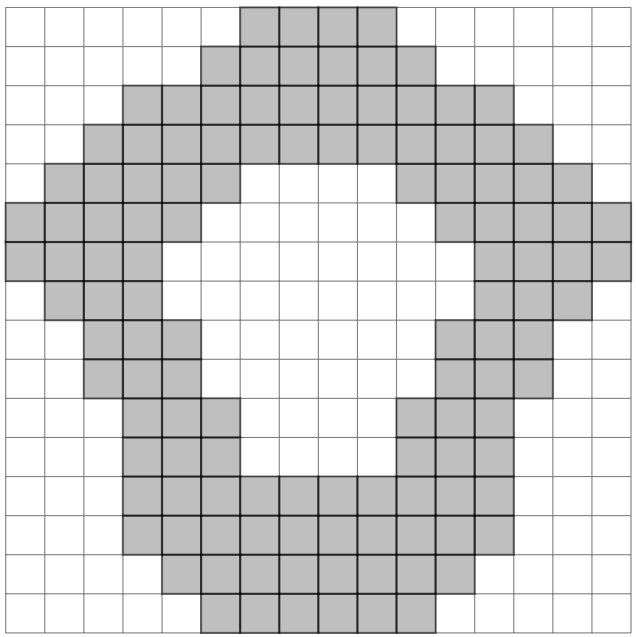
We transform the input grid. Blue contour is the object's contour in the target image and the gray area is the transformation of the moving image. So, everything works nicely.

However, here everything is continuous and analytically computed! Actually this is not how it would work in discrete case.

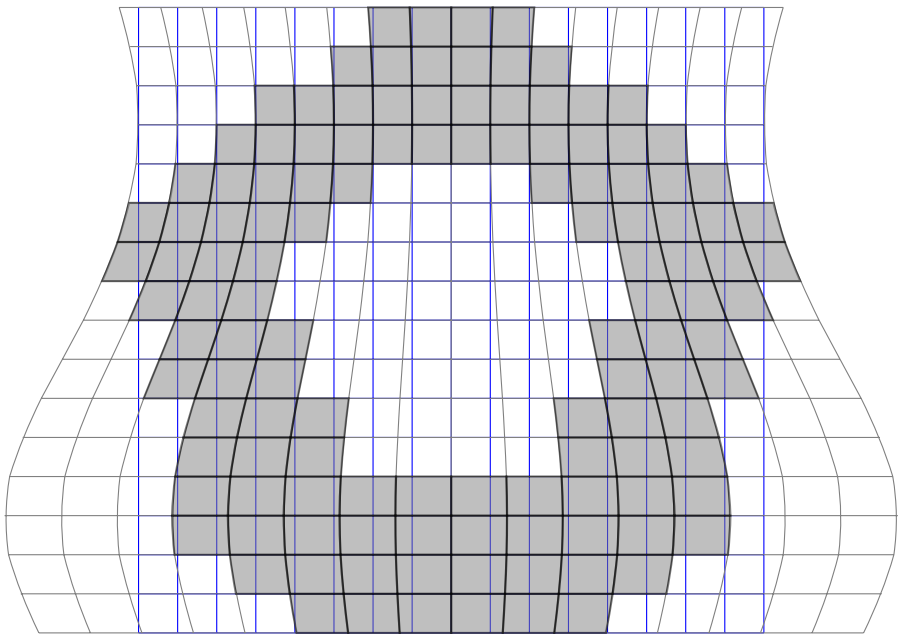
Forward transformation: discrete case

Let us use the following forward transformation / mapping

$$T(x) = \{x_1 / [0.15 \sin(2.5 x_2) + 0.85], x_2\} = \{x_1', x_2'\} = x'$$



Input grid and Moving image

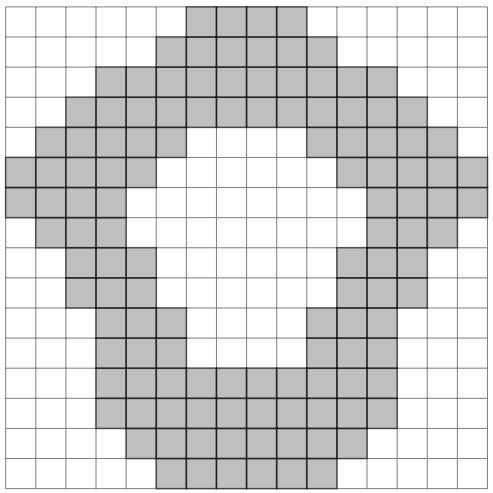


Output grid and Transformed grid

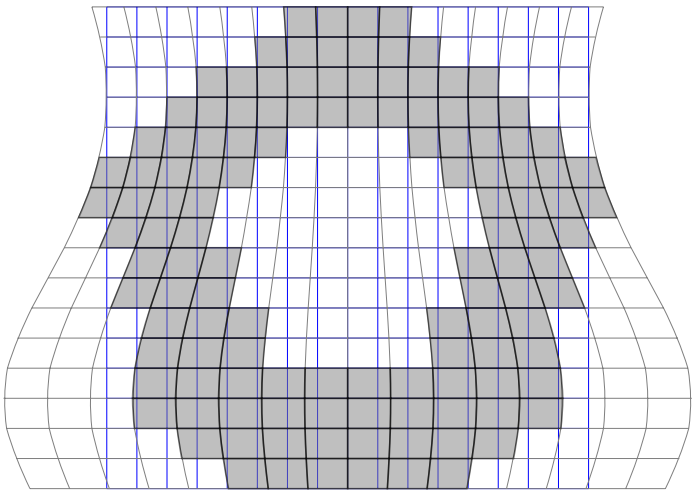
In the discrete case each pixel from the moving image will be transformed $x' = T(x)$ and the pixel intensity will be copied to the output image.
This approach is problematic!

Forward transformation is problematic

$$T(x) = \{x_1 / [0.15 \sin(2.5 x_2) + 0.85], x_2\} = \{x_1', x_2'\} = x'$$



Input grid and Moving image



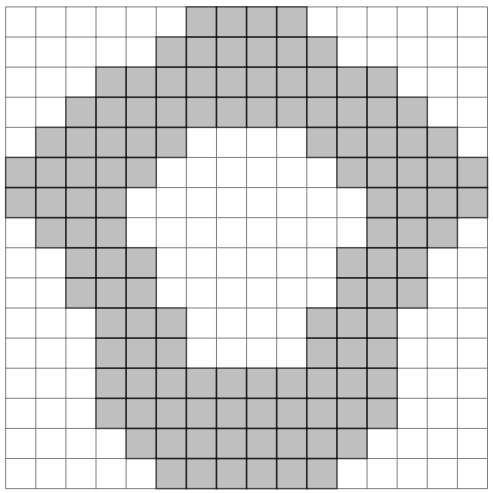
Output grid and Transformed grid

Problems:

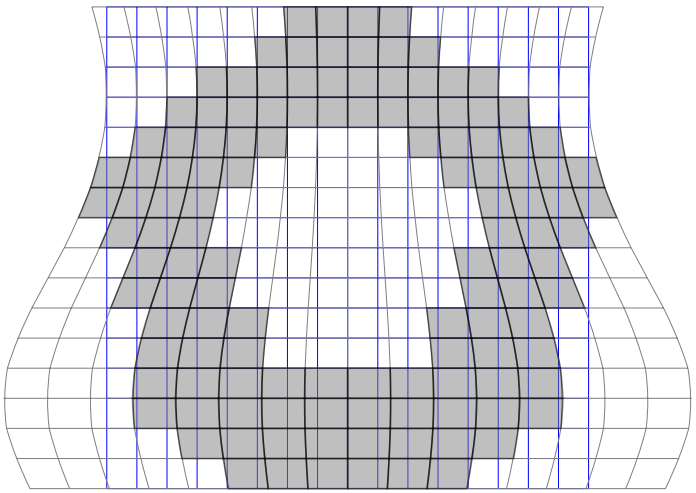
- Transformed pixel coordinates x' does not necessarily fall on pixel exactly.
- Two pixels can be mapped to the same output pixel.
- Same output pixel can be mapped to two output pixels.
- Some output pixels may not even get a value assigned.

Forward transformation is problematic

$$T(x) = \{x_1 / [0.15 \sin(2.5x_2) + 0.85], x_2\} = \{x_1', x_2'\} = x'$$



Input grid and Moving image



Output grid and Transformed grid

Problems:

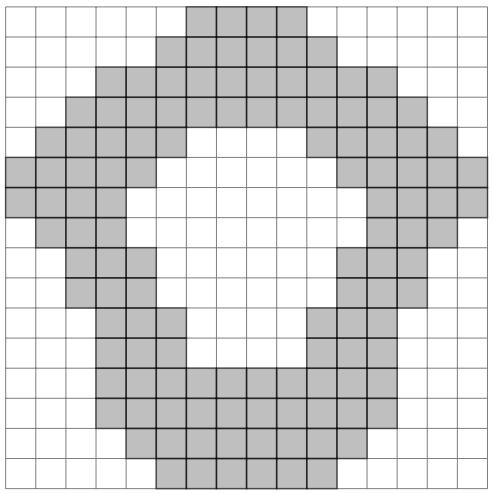
- Transformed pixel coordinates x' does not necessarily fall on pixel exactly.
- Two pixels can be mapped to the same output pixel.
- Same output pixel can be mapped to two output pixels.
- Some output pixels may not even get a value assigned.

Solution:

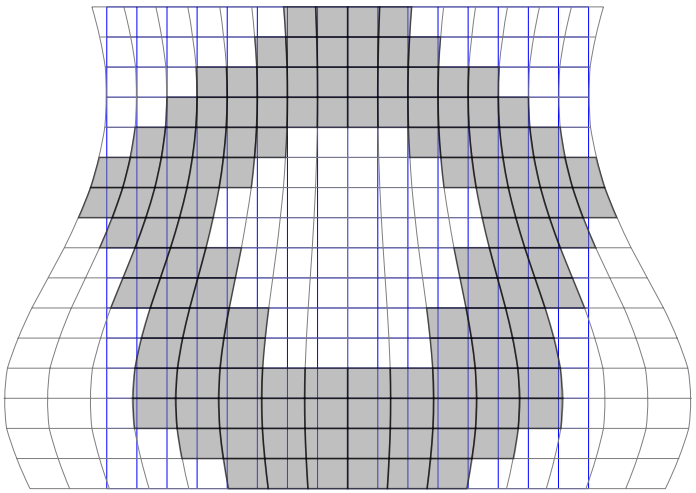
- Intensities of the output pixels can be computed using partial areas.
- Overlap between each transformed pixel and output pixels are computed.
- This overlap is used in a complex intensity assignment step.

Forward transformation is problematic

$$T(x) = \{x_1 / [0.15 \sin(2.5x_2) + 0.85], x_2\} = \{x_1', x_2'\} = x'$$



Input grid and Moving image



Output grid and Transformed grid

Problems:

- Transformed pixel coordinates x' does not necessarily fall on pixel exactly.
- Two pixels can be mapped to the same output pixel.
- Same output pixel can be mapped to two output pixels.
- Some output pixels may not even get a value assigned.

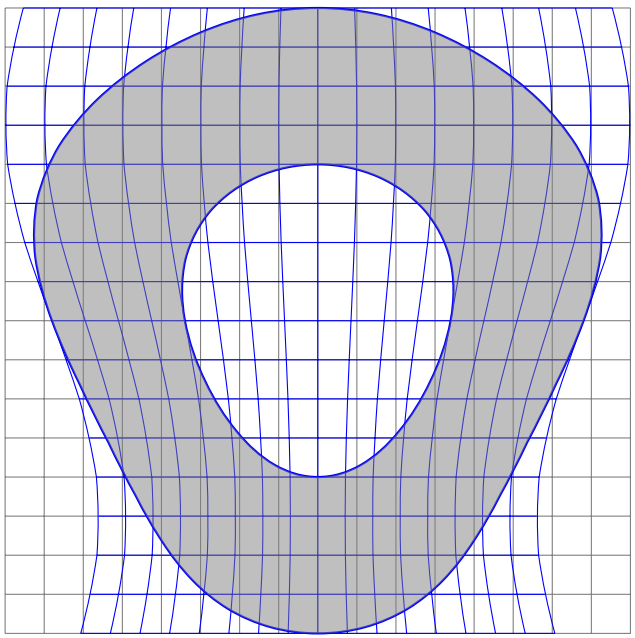
Solution: Quite complicated

- Intensities of the output pixels can be computed using partial areas.
- Overlap between each transformed pixel and output pixels are computed.
- This overlap is used in a complex intensity assignment step.

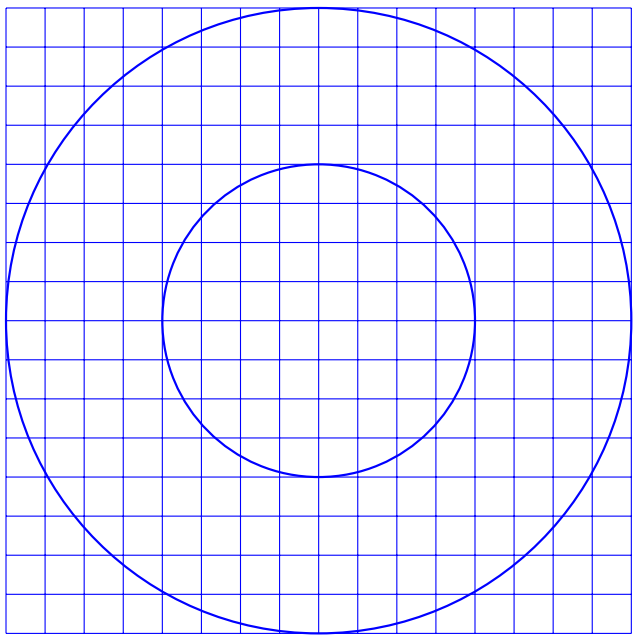
Backward transformation

Let us use the following inverse transformation / mapping:

$$T^{-1}(x') = \left\{x_1' \left[0.15 \sin(2.5 x_2) + 0.85\right], x_2'\right\} = \{x_1, x_2\} = x$$



Input and transformed grid



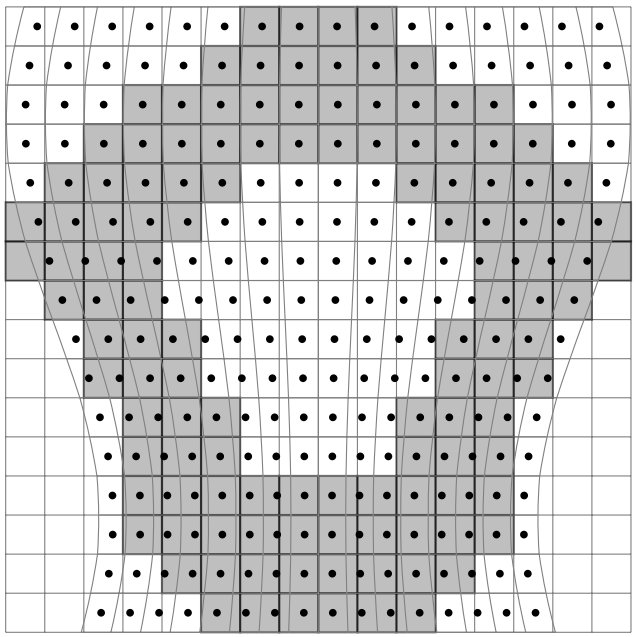
Output grid

We transform the output grid. Blue contour is the object's contour in the target image and the gray are is the transformation of the moving image. Everything is continuous and analytical again.

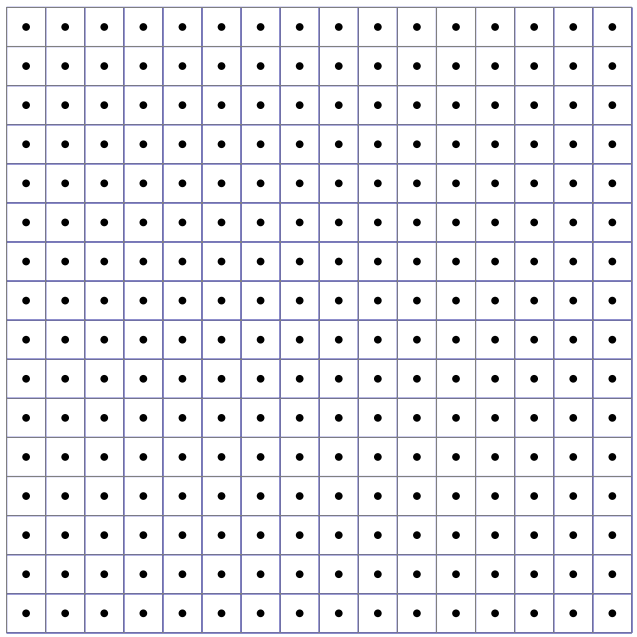
Backward transformation: discrete case

Let us use the following inverse transformation / mapping:

$$T^{-1}(x') = \left\{x_1' \left[0.15 \sin(2.5 x_2) + 0.85\right], x_2'\right\} = \{x_1, x_2\} = x$$



Input grid / Moving image



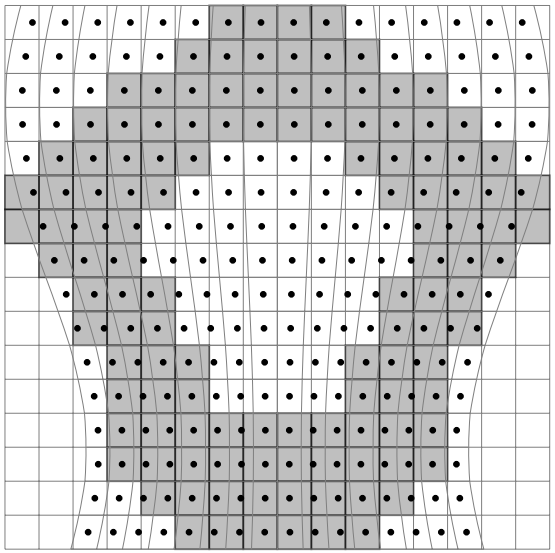
Output grid / Output image

In the backward mapping, for each output pixel the corresponding location in the input / moving image is determined. This is much easier to deal with.

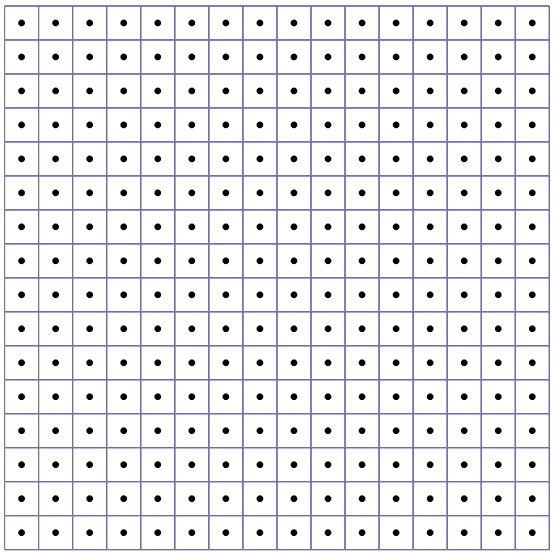
Backward transformation is easier to use

Let us use the following inverse transformation / mapping:

$$T^{-1}(x') = \left\{x_1' \left[0.15 \sin(2.5 x_2) + 0.85\right], x_2'\right\} = \{x_1, x_2\} = x$$



Input grid / Moving image



Output grid / Output image

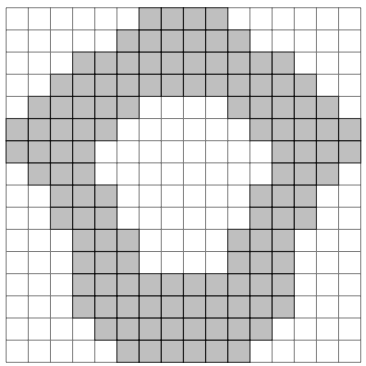
Problems:

- Transformed locations of the output pixels do not generally fall exacty on input pixels.

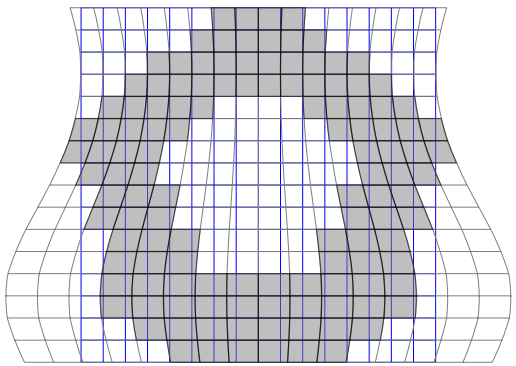
Solution:

- Interpolation
- Pixel intensity (all channels if necessary) are interpolated using the nearest input image pixels.

Summary

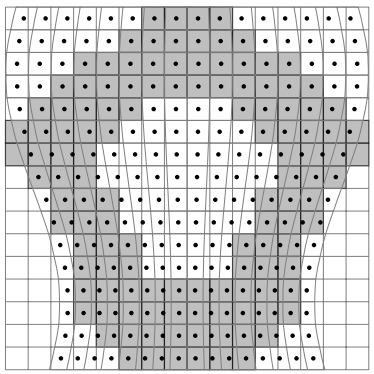


Input grid and Moving image

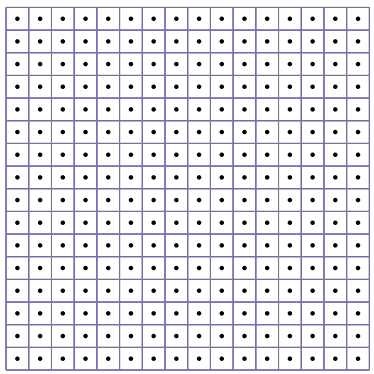


Output grid and Transformed grid

Forward transformation



Input grid / Moving image



Output grid / Output image

Backward transformation

Keep in mind

Backward transformation is always easier to use for transforming images through interpolation.

Section 3

Interpolation models

Outline

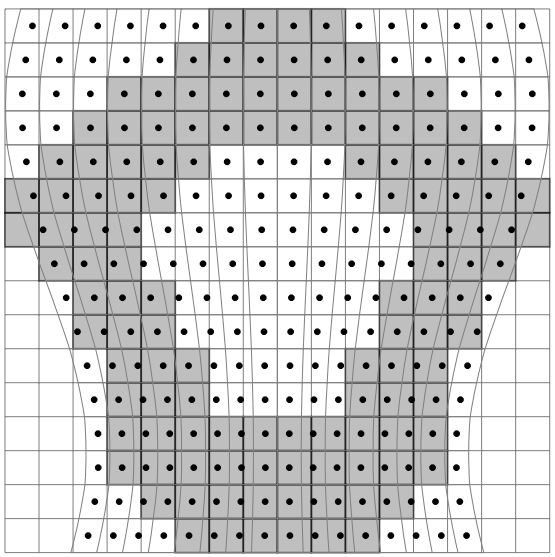
- Sampling model - How to apply a transformation
- Interpolation models
  - Nearest-neighbor interpolation
  - Bilinear interpolation
  - Cubic interpolation
- Linear transformation models
- Non-linear transformation models

Interpolation is used extensively

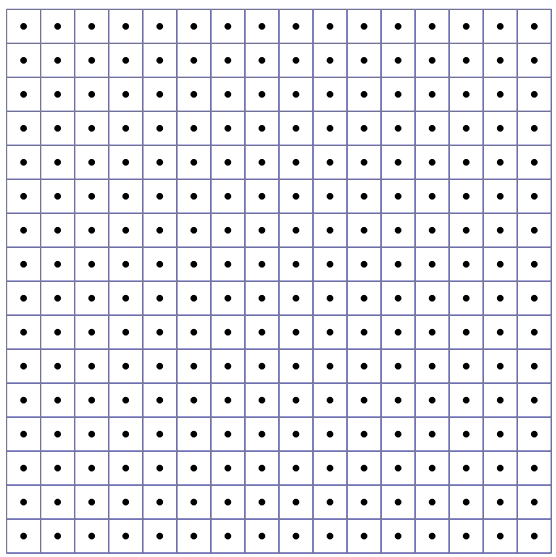
When is it used?

Whenever an image is resampled interpolation is used to compute the intensities at the new pixels.

Example I: Spatial transformations



Input grid / Moving image



Output grid / Output image

Interpolation is used extensively

When is it used?

Whenever an image is resampled interpolation is used to compute the intensities at the new pixels.

Example II: Resizing an image for pre-processing
Images of the same person one year apart
Sizes of the images on the slides are proportional to their number of pixels.

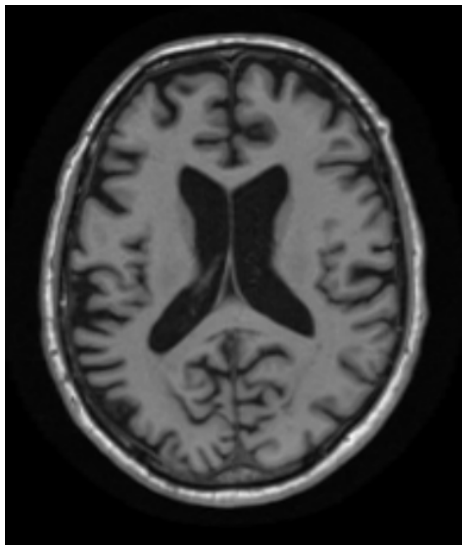


Image size: [454, 384]
Pixel size: [1.25, 1.25]

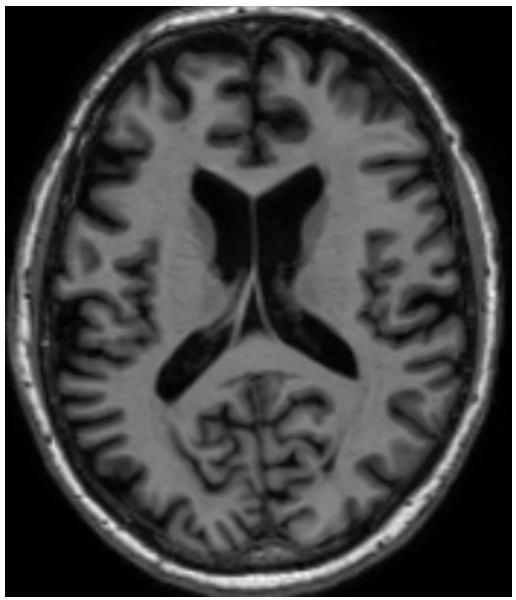


Image size: [494, 424]
Pixel size: [1.0, 1.0]

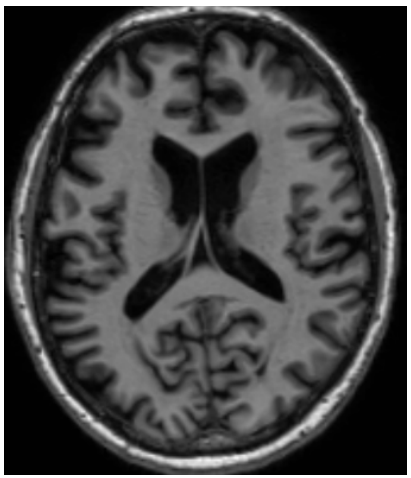
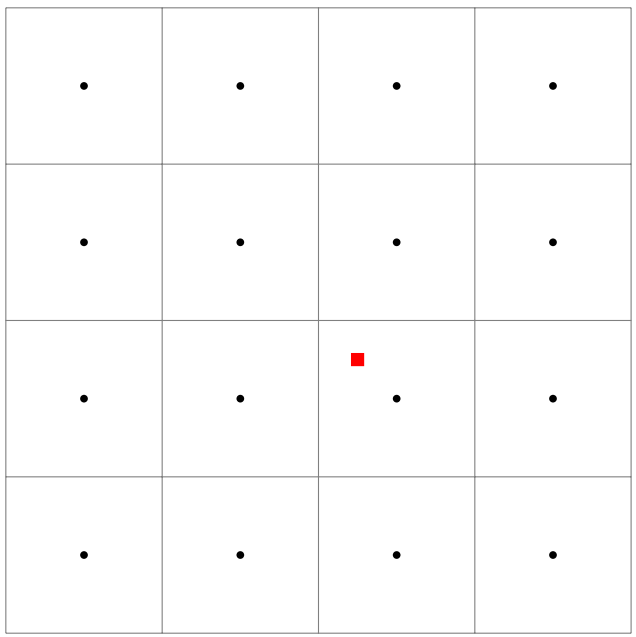


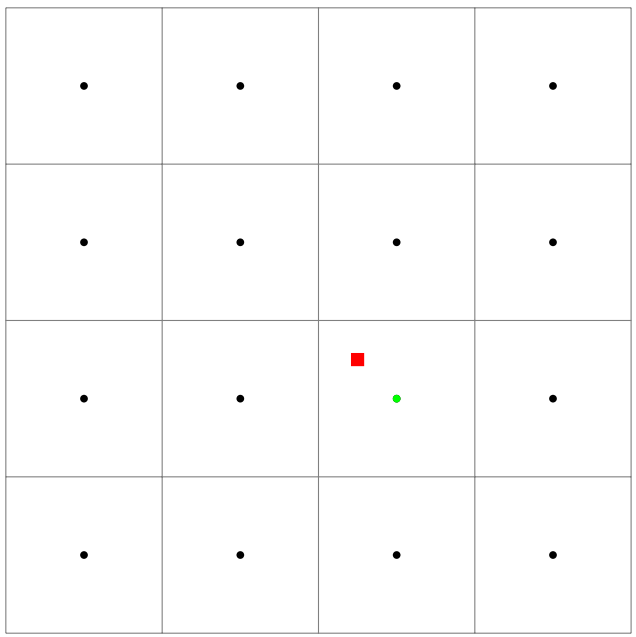
Image size: [395, 339]
Pixel size: [1.25, 1.25]

Problem setup



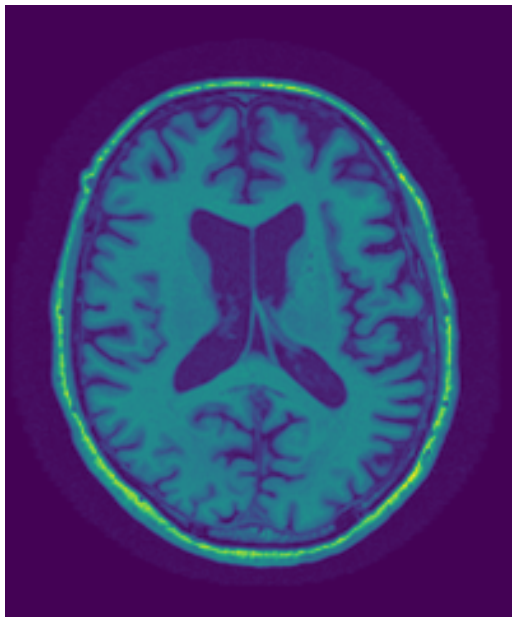
Intensities at each pixel are defined and we assume those intensities are defined at the center of the pixels, black nodes. How can we compute the intensity at the red node?

Nearest neighbor interpolation

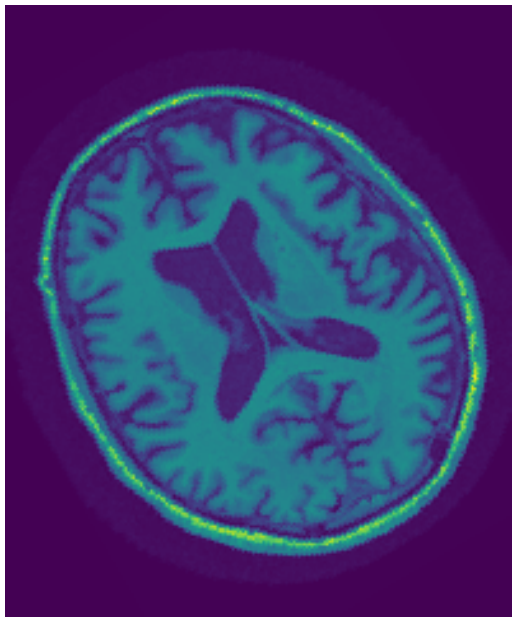


Assign the intensity value of the closest pixel with known intensity.
Assign the value of the "nearest neighbor".

Nearest neighbor interpolation on images



Original



Rotated by 30 degrees

- Nearest neighbor interpolation is very fast.
- It does not introduce new intensity values in the image.
- May introduce heavy artifacts in gray level images. [See boundaries of the object]
- Nearest neighbor interpolation yields a piece-wise constant function. This is not continuous nor differentiable.

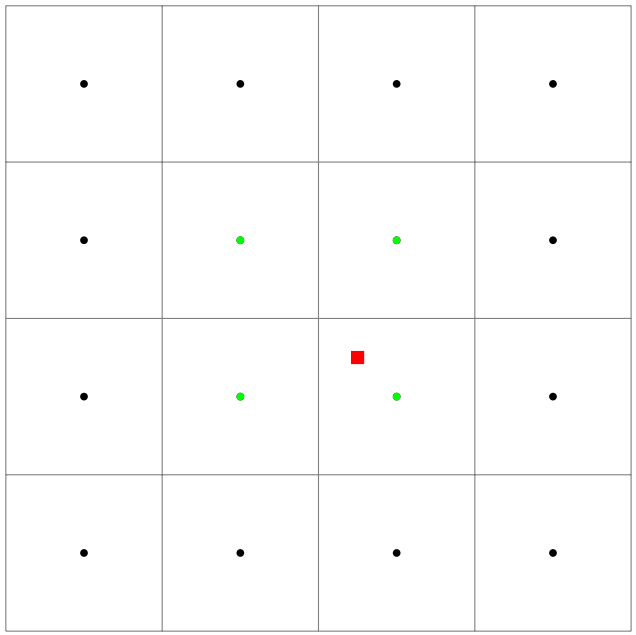
Nearest neighbor interpolation on label maps

Original

Rotated by 30 degrees

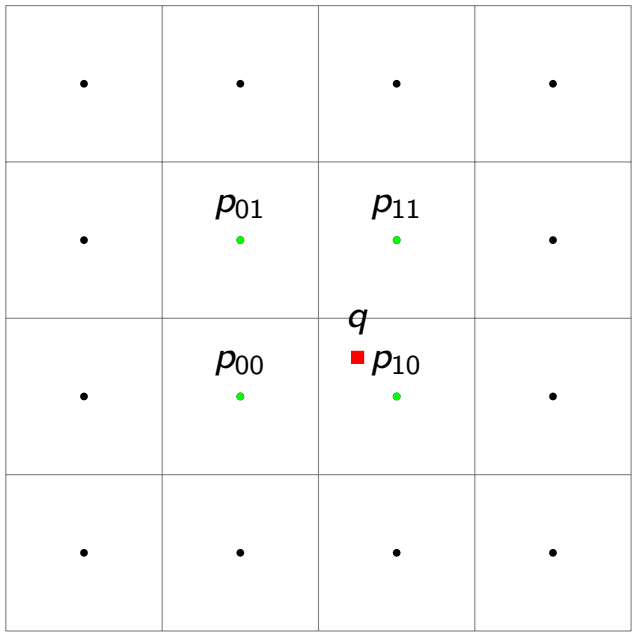
While it is not used for gray level or multispectral images, it is used for transforming labels since it has the nice property of not introducing any new values.

Bilinear interpolation



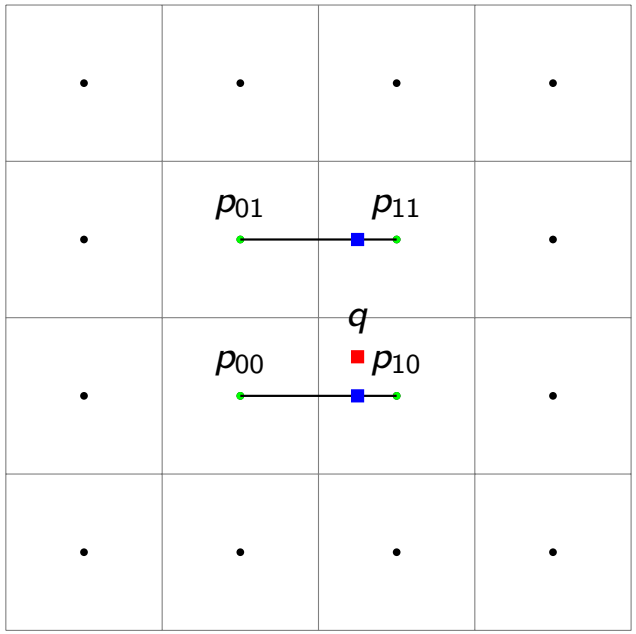
- Uses closest four pixel's intensity values.
- Extension of linear interpolation in 2D.
- Trilinear interpolation in 3D.

Bilinear interpolation - linear interpolation in one dimension



Let us denote the coordinates of the four points as $p_{00} = \{x_1^0, x_2^0\}$, $p_{01} = \{x_1^0, x_2^1\}$, $p_{10} = \{x_1^1, x_2^0\}$, $p_{11} = \{x_1^1, x_2^1\}$ and $q = \{x_1, x_2\}$.

Bilinear interpolation - linear interpolation in one dimension

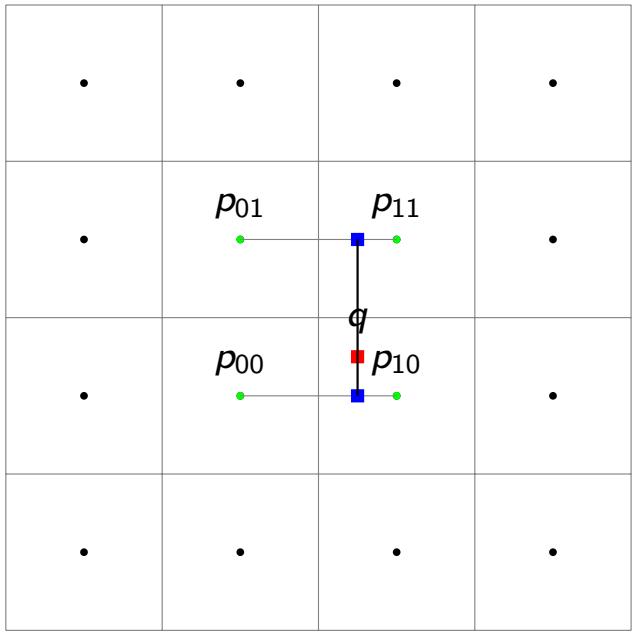


Let us denote the coordinates of the four points as $p_{00} = \{x_1^0, x_2^0\}$, $p_{01} = \{x_1^0, x_2^1\}$, $p_{10} = \{x_1^1, x_2^0\}$, $p_{11} = \{x_1^1, x_2^1\}$ and $q = \{x_1, x_2\}$. Interpolated values at the blue intermediate points are:

$$I(x_1, x_2^1) = \frac{x_1^1 - x_1}{x_1^1 - x_1^0} I(x_1^0, x_2^1) + \frac{x_1 - x_1^0}{x_1^1 - x_1^0} I(x_1^1, x_2^1)$$

$$I(x_1, x_2^0) = \frac{x_1^1 - x_1}{x_1^1 - x_1^0} I(x_1^0, x_2^0) + \frac{x_1 - x_1^0}{x_1^1 - x_1^0} I(x_1^1, x_2^0)$$

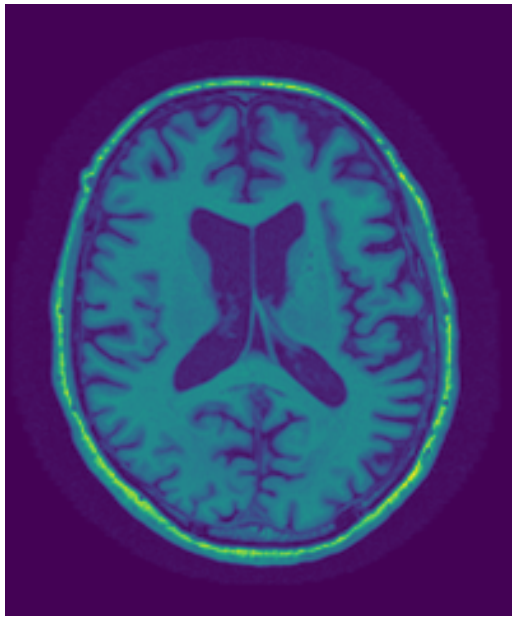
Bilinear interpolation - linear interpolation in the other dimension



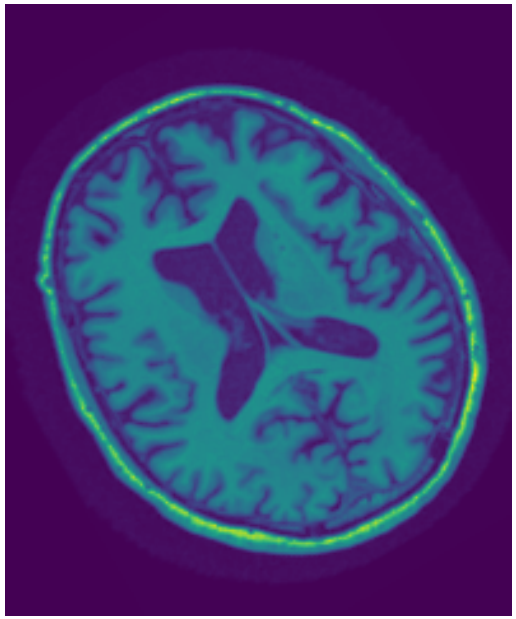
Let us denote the coordinates of the four points as $p_{00} = \{x_1^0, x_2^0\}$, $p_{01} = \{x_1^0, x_2^1\}$, $p_{10} = \{x_1^1, x_2^0\}$, $p_{11} = \{x_1^1, x_2^1\}$ and $q = \{x_1, x_2\}$. Interpolated values at the blue intermediate points are:

$$I(x_1, x_2) = \frac{x_2^1 - x_2}{x_2^1 - x_2^0} I(x_1, x_2^0) + \frac{x_2 - x_2^0}{x_2^1 - x_2^0} I(x_1, x_2^1)$$

Bilinear interpolation on images



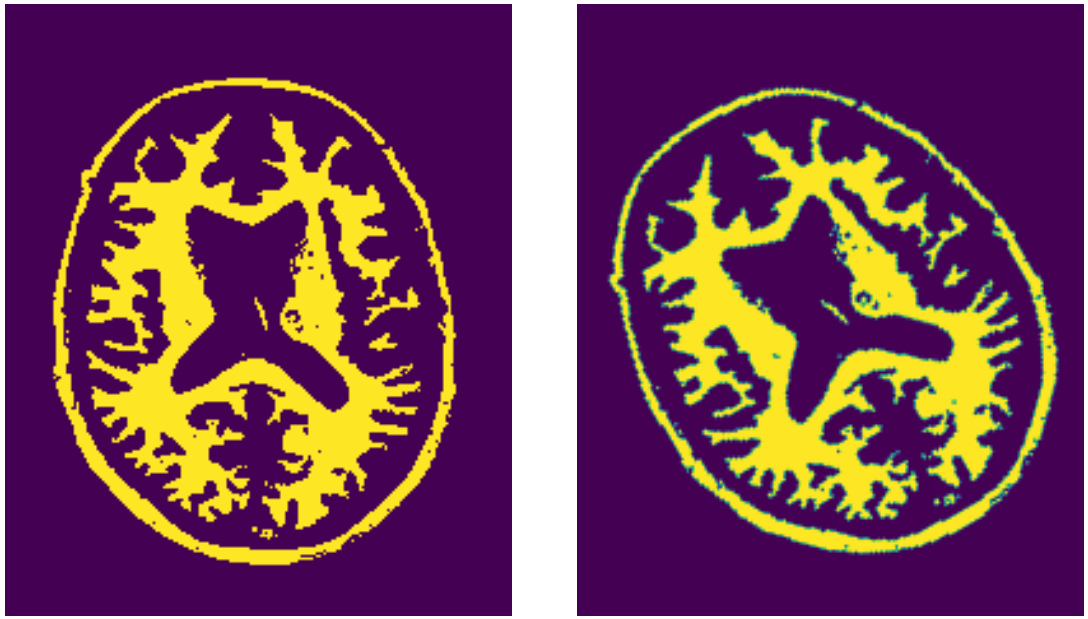
Original



Rotated by 30 degrees

- Bilinear interpolation is fast.
- It will introduce new intensity values in the image due to interpolation
- Does not introduce as big artifacts as nearest neighbor interpolation.
- Bilinear interpolation leads to piece-wise linear functions, which is continuous but not differentiable.
- It may smooth and blur the image, leading to decrease in spatial resolution.
- Trilinear interpolation in 3D repeats the same procedure for an additional dimension and uses 8 points - corners of a cube.

Bilinear interpolation on on label maps



Original | Rotated by 30 degrees

- On label maps bilinear interpolation produces extra values [look at the boundaries].
- These additional labels arise due to interpolation.
- They do not exist and should be avoided whenever resampling label maps.

Bilinear interpolation - alternative formulation

The bilinear interpolation has the following form with four unknown parameters.

$$I(x_1, x_2) = a + bx_1 + cx_2 + dx_1x_2$$

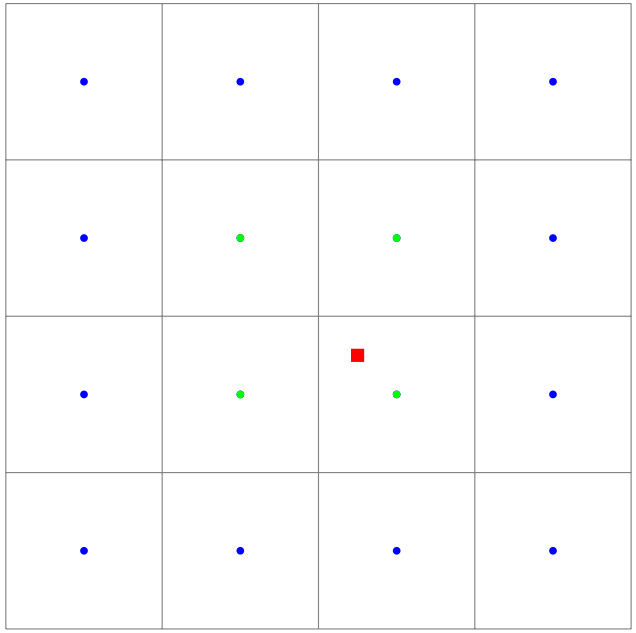
Corresponding to the four unknowns we have the intensity values at the four corners leading to a linear system of equation:

$$\begin{bmatrix} 1 & x_1^0 & x_2^0 & x_1^0x_2^0 \\ 1 & x_1^0 & x_2^1 & x_1^0x_2^1 \\ 1 & x_1^1 & x_2^0 & x_1^1x_2^0 \\ 1 & x_1^1 & x_2^1 & x_1^1x_2^1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} I(x_1^0, x_2^0) \\ I(x_1^0, x_2^1) \\ I(x_1^1, x_2^0) \\ I(x_1^1, x_2^1) \end{bmatrix}$$

One can solve the system of equations and use a, b, c, d coefficients to interpolation at any x in between $p_{00}, p_{01}, p_{10}, p_{11}$.
Trilinear interpolation can be defined similarly with the following parametric form with 8 unknowns

$$I(x_1, x_2, x_3) = a + bx_1 + cx_2 + dx_3 + ex_1x_2 + fx_1x_3 + gx_2x_3 + hx_1x_2x_3$$

Bicubic interpolation



- Uses closest four pixel's information.
- However, it requires not only the intensity values but also the derivates at those points.
- Therefore, it is common to use 16 closest points in the interpolation.
- Extension of cubic interpolation in 2D.
- Tricubic interpolation in 3D.
- Similar idea as bilinear interpolation: cubic interpolation in one dimension followed by the other.

Bicubic interpolation - functional form

Bicubic interpolation has the following form with 16 unknowns:

$$I(x_1, x_2) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x_1^i x_2^j$$

Similar to the bilinear interpolation, a linear system of equations can be constructed and solved. Information to determine these unknowns are taken from the four closest neighbors as:

- Intensities at $p_{00}, p_{01}, p_{10}, p_{11} = 4$ values.
- ∂x_1 and ∂x_2 derivatives at $p_{00}, p_{01}, p_{10}, p_{11} = 8$ values.
- $\partial x_1 x_2$ mixed derivatives at $p_{00}, p_{01}, p_{10}, p_{11} = 4$ values.

Remark

In order to compute the derivatives further closest pixels must be used. Therefore, bicubic interpolation uses more pixels than the closest 4.

Remark

Approximating the derivatives can be done in different ways.

Bicubic interpolation - convolutional form

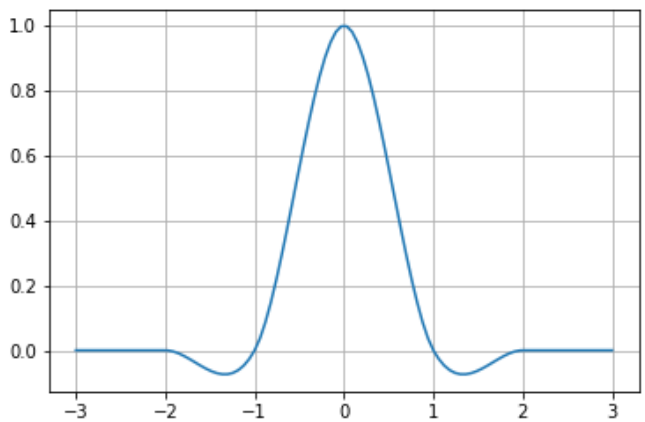
[R. Keys (1981). "Cubic convolution interpolation for digital image processing"]

One easy way to do bicubic interpolation is through convolution with the following kernel - a result by Keys:

$$K(x) = \begin{cases} \frac{3}{2}|x|^3 - \frac{5}{2}|x|^2 + 1, & 0 \leq |x| \leq 1 \\ -\frac{1}{2}|x|^3 + \frac{5}{2}|x|^2 - 4|x| + 2, & 1 \leq |x| \leq 2 \\ 0, & 2 \leq |x|. \end{cases}$$

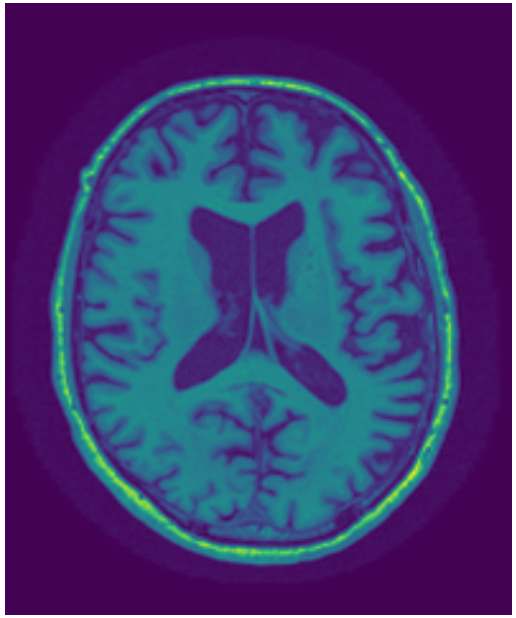
$$f(x) = \sum_{i=-2}^{2} f(x^i)K(x - x^i)$$

- The kernel is used to convolve each dimension one after the other.
- In 1d its support is in $[-2, 2]$ taking into account 4 sampled points.
- in 2d the bicubic interpolation with Keys' convolution takes into account 4x4 closest pixels to the interpolating point x.
- The kernel is called "Catmull-Rom" kernel.

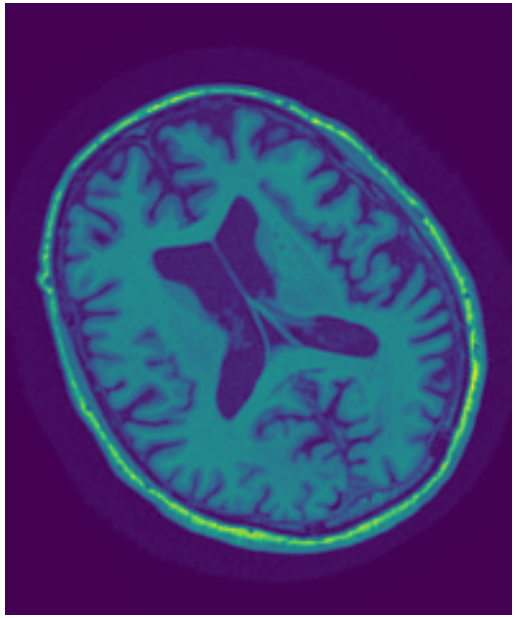


Catmull-Rom kernel

Bicubic interpolation on images



Original



Rotated by 30 degrees

- Bicubic interpolation is better at retaining gray values than bilinear and nearest neighbor. It has the highest quality interpolation amongst them.
- Computational requirement is the highest.
- Bicubic interpolation leads to smooth functions that are both continuous and differentiable.
- Tricubic interpolation in 3D repeats the same procedure for an additional dimension and larger number of points.

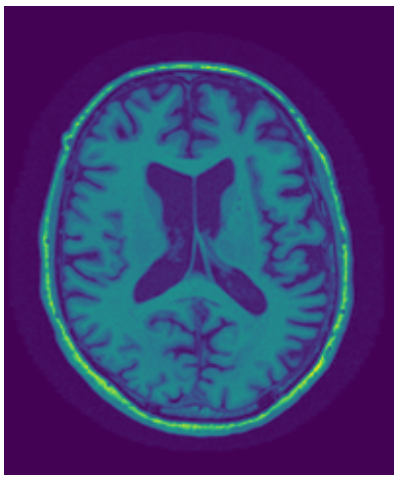
Bicubic interpolation on on label maps

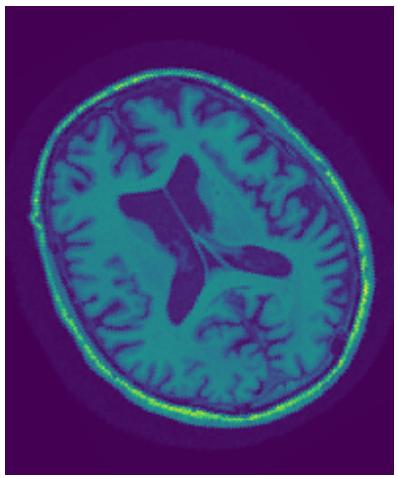
Original

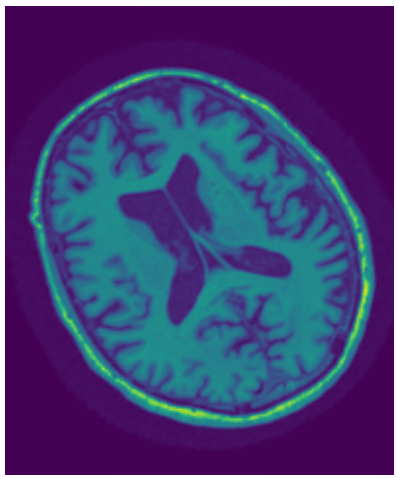
Rotated by 30 degrees

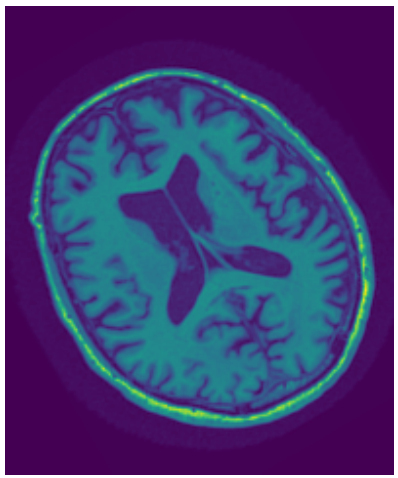
- On label maps bicubic interpolation produces extra values [look at the boundaries].
- These additional labels arise due to interpolation.
- They do not exist and should be avoided whenever resampling label maps.

Comparisons


Original

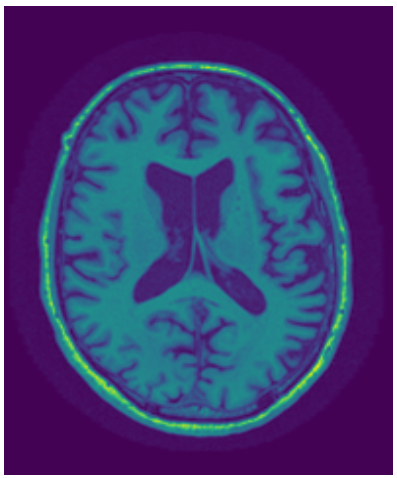

Nearest neighbor


Bilinear

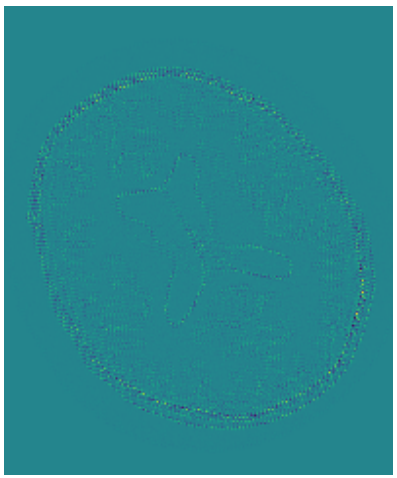

Bicubic

- Nearest neighbor yields bad quality.
- Bilinear is a good trade-off.
- During a registration, the interpolation needs to be computed many times during the optimization. Bilinear's computational advantage is important for this.
- One may use bilinear during the optimization but bicubic at the end.
- For transforming label maps, it is better to use nearest neighbor interpolation.

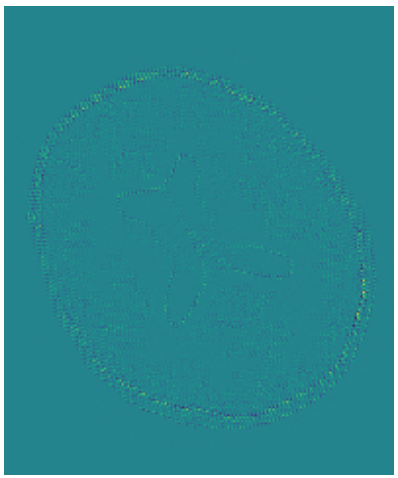
Comparisons



Original



Bilinear - NN

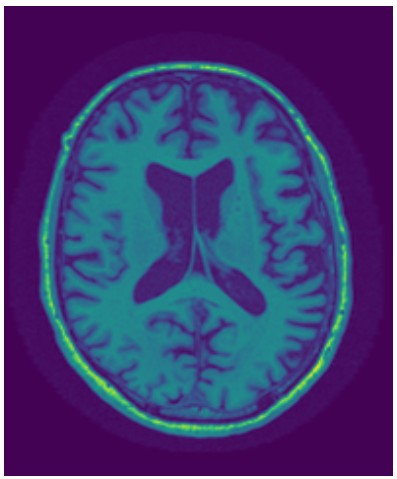


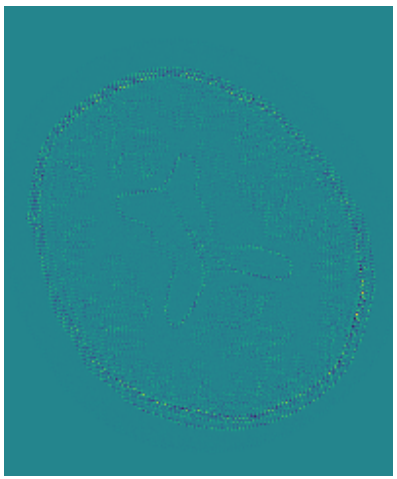
Bicubic - NN

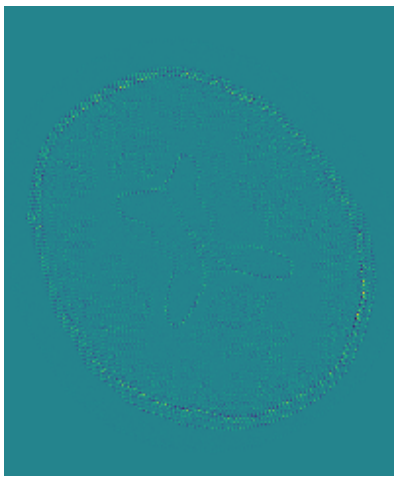
Bicubic - Bilinear

- Nearest neighbor yields bad quality.
- Bilinear is a good trade-off.
- During a registration, the interpolation needs to be computed many times during the optimization. Bilinear's computational advantage is important for this.
- One may use bilinear during the optimization but bicubic at the end.
- For transforming label maps, it is better to use nearest neighbor interpolation.

Comparisons


Original


Bilinear - NN


Bicubic - NN


Bicubic - Bilinear

A word on kernels

$$f(x) = \sum_{i=-N}^{N} f(x^i) K(x - x^i)$$

The above form is very generic and one may consider different kernels for the same task.

Outline

- Sampling model - How to apply a transformation
- Interpolation models
- Linear transformation models
- Non-linear transformation models