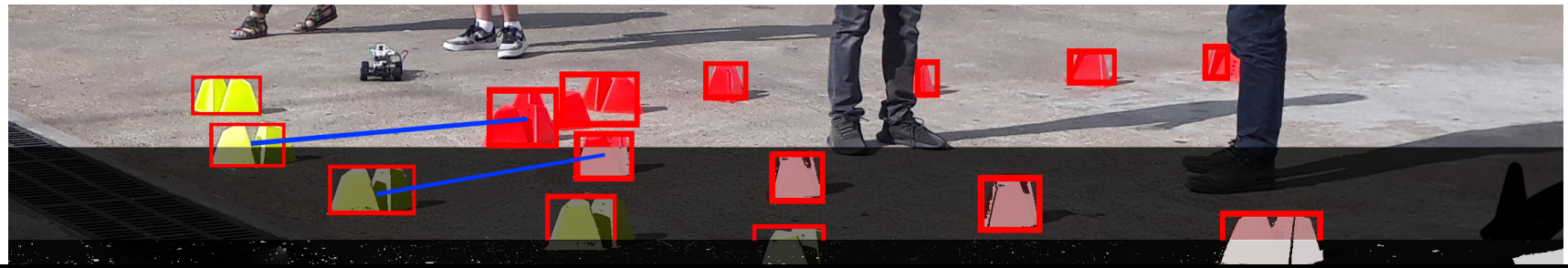
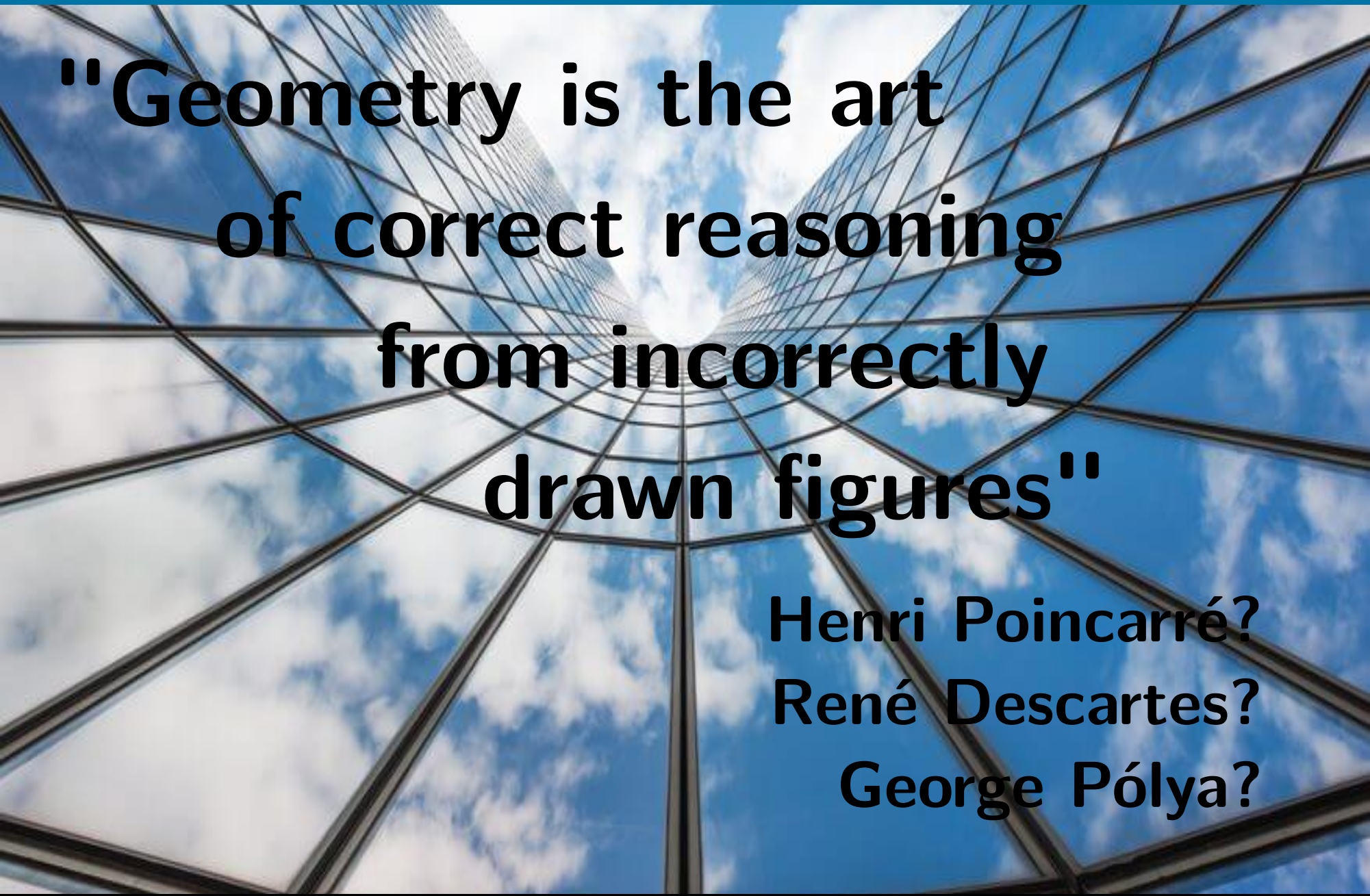
[IPRO] Computer vision – Preparing and improving (I) Geometric transformations

Explorers' journey – Second Camp

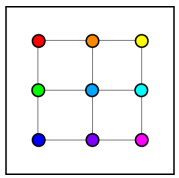
Laurent Beaudoin & Loïca Avanthey

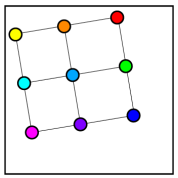
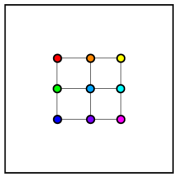
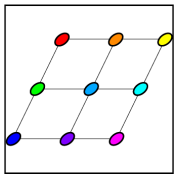
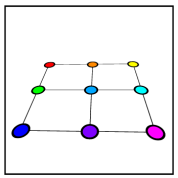



"Geometry is the art
of correct reasoning
from incorrectly
drawn figures"
Henri Poincarré?
René Descartes?
George Pólya?

Geometric transformation? Very very distant reminders

Geometric transformation *"It is a transformation that changes the* ***position of pixels*** *(permutation)"*



	**Isometries** (reflections, translations, rotations)	Preserve distances and angles
	**Similarities** (isometries + homotheties)	Preserve angles and distance ratios
	**Affinities** (similarities + affine transf.)	Preserve parallelism
	**Homographies** (affinities + projective transf.)	Preserve lines (collinearity)

Reflections (isometries)

During the acquisition the sensor can be **upside down**, so we need to **reverse** the images (horizontal flip or reflection)

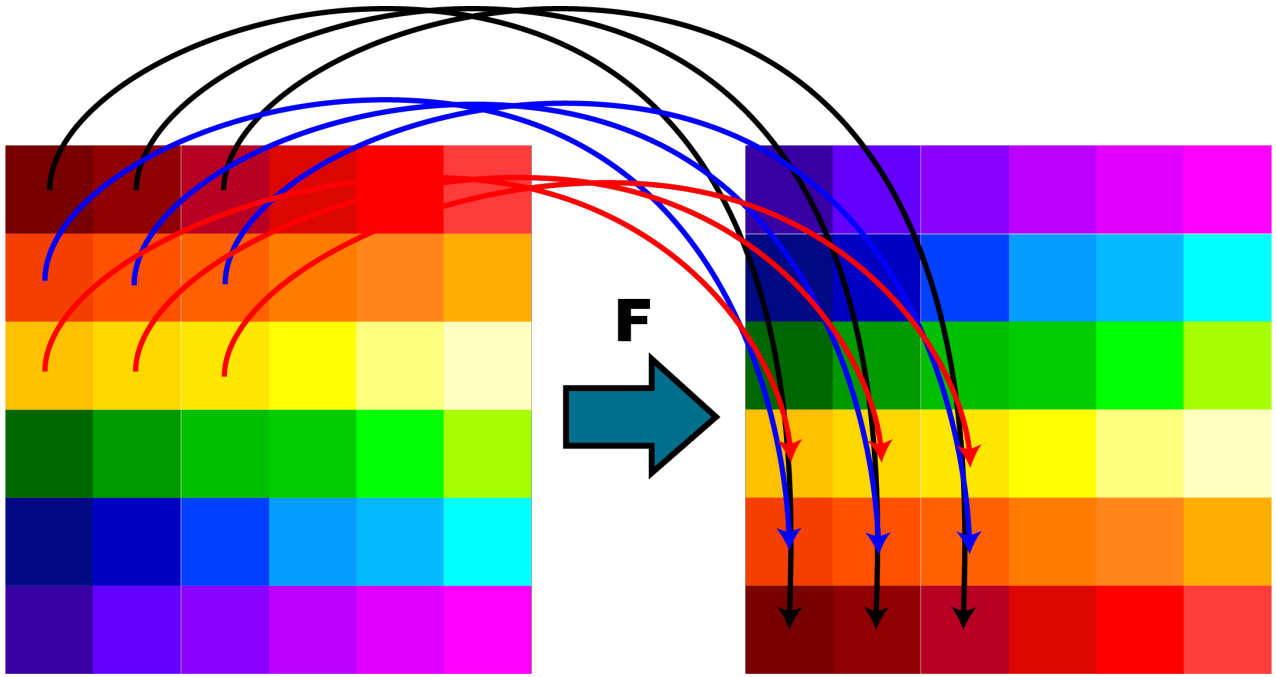

[CC BY NC SA L. Beaudoin et al., 2018]

The **other reflection** (vertical flip) is mostly used in **photography** or **infography** (esthetics) and in **cinematography** (obfuscation of texts and logos)

Reflections: how to?

We put the pixels of the **first line** on the **last line**, the **second** on the **last before** and so on

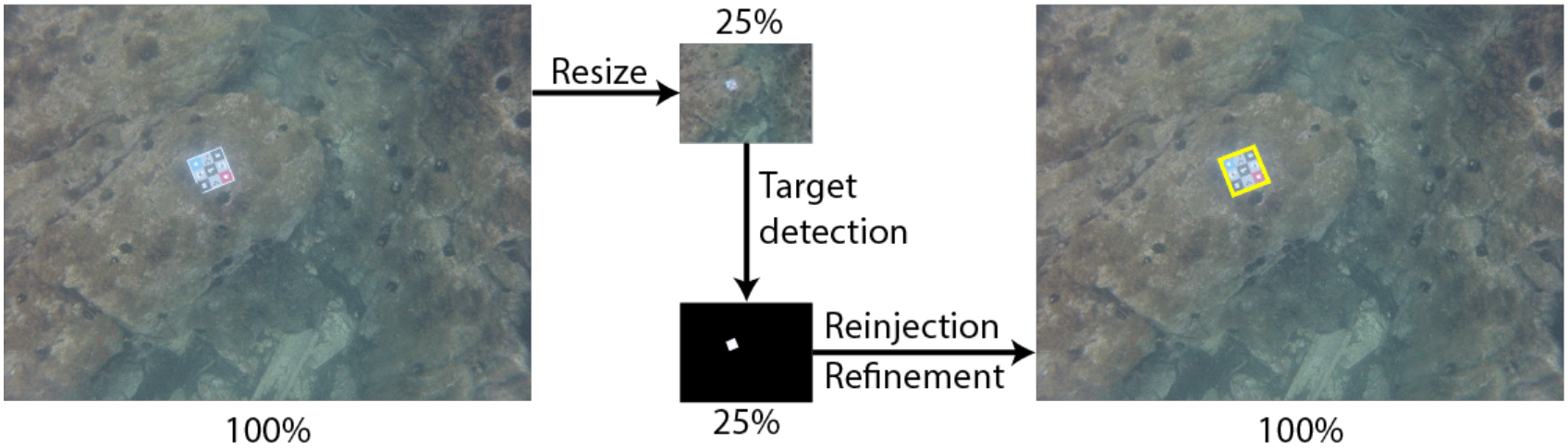
$$F(x,y): \begin{cases} x & \mapsto & x \\ y & \mapsto & height - 1 - y \end{cases} \quad \forall x \in \mathbb{N},\ \forall y \in \mathbb{N}$$



[CC BY L. Avanthey et al., 2018]

Homotheties (similarities) (1)

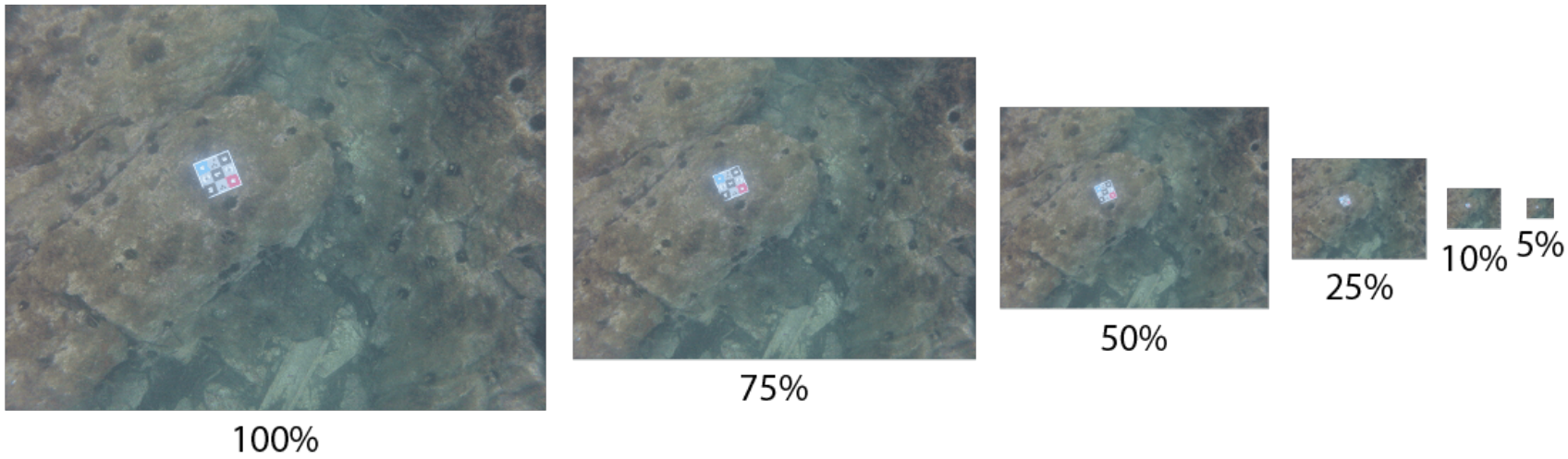
To **speed up** (quantity) or **facilitate** (detail) the **processing**, we need to **change the size** of the images (subsampling)



[L. Beaudoin et al., 2018]

Homotheties (similarities) (2)

Sometimes, for the reinjection of details or image compression for example, we work on **several sizes at once**: we call this a **pyramid of resolutions**

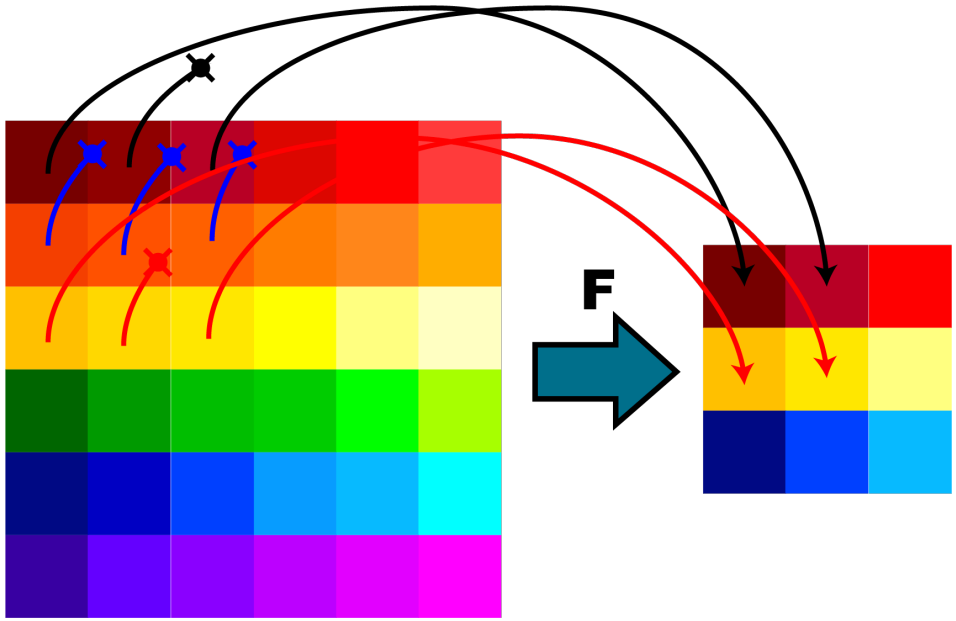


[L. Avanthey et al., 2018]

Resize: how to ? (simple case) (1)

The **final size** is a **divisor** of the **initial one** (eg. an even height & width and a scale factor of 50% ie $S = 2$)

$$F(x,y) : \begin{cases} x & \mapsto & x/S \\ y & \mapsto & y/S \end{cases} \quad \forall S \mid (width, height) \equiv 0[S],\ \forall x \in \mathbb{N},\ \forall y \in \mathbb{N}$$



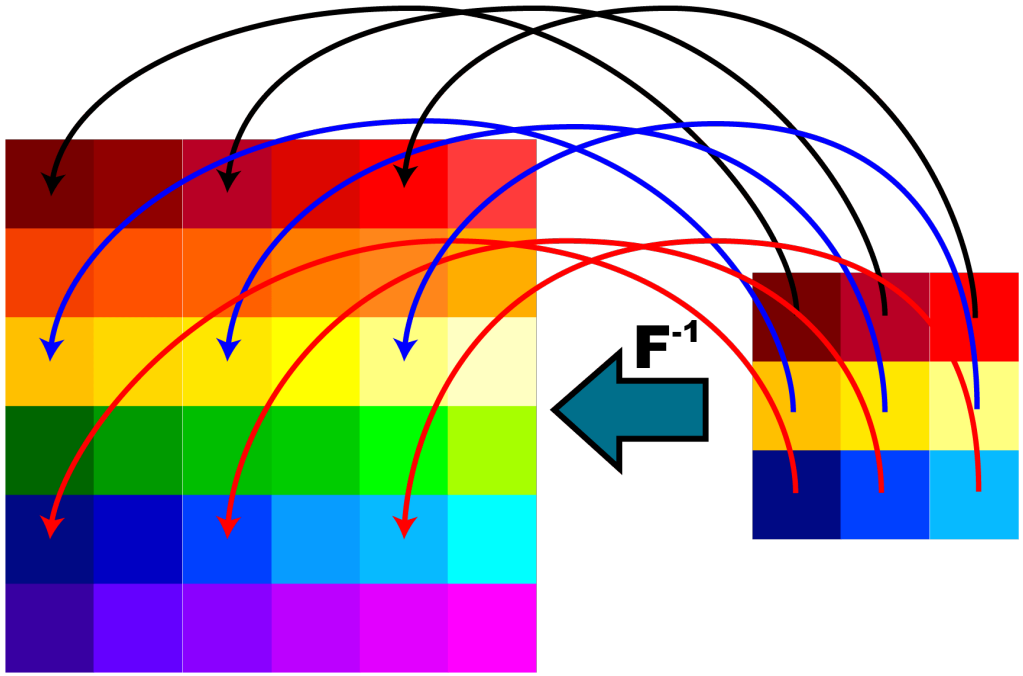
[CC BY L. Avanthey et al., 2018]

Can we be **more efficient**?

Resize: how to ? (simple case) (2)

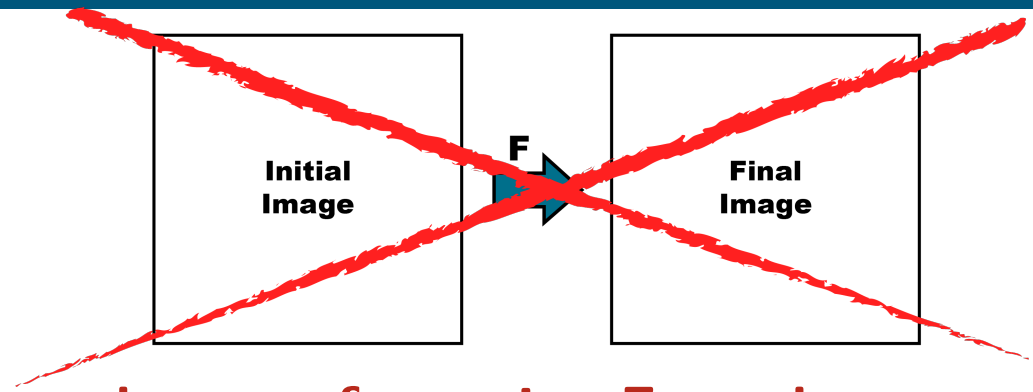
If we start from the **final image** and apply the **inverse transformation** to know which pixel to take in the **initial image** then the number of iterations is divided by S^2

$$F^{-1}(x,y) : \begin{cases} x & \mapsto & x * S \\ y & \mapsto & y * S \end{cases} \quad \forall S \mid (width, height) \equiv 0[S],\ \forall x \in \mathbb{N},\ \forall y \in \mathbb{N}$$

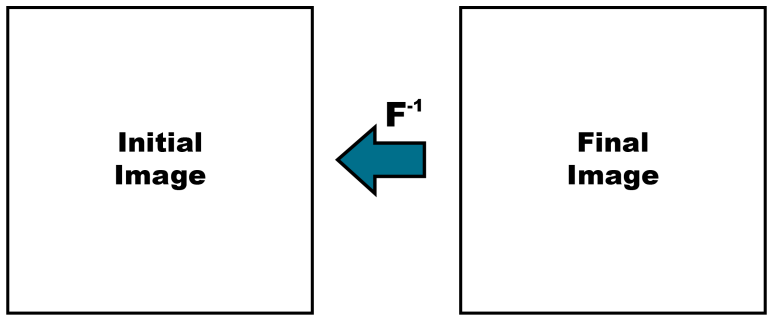


[L. Avanthey et al., 2018]

Transformations: initial image, final image



Whatever the transformation F, we **do not use** the strategy which consists for each pixel of the **initial image** to find the corresponding pixel on the **final image** according to **F**

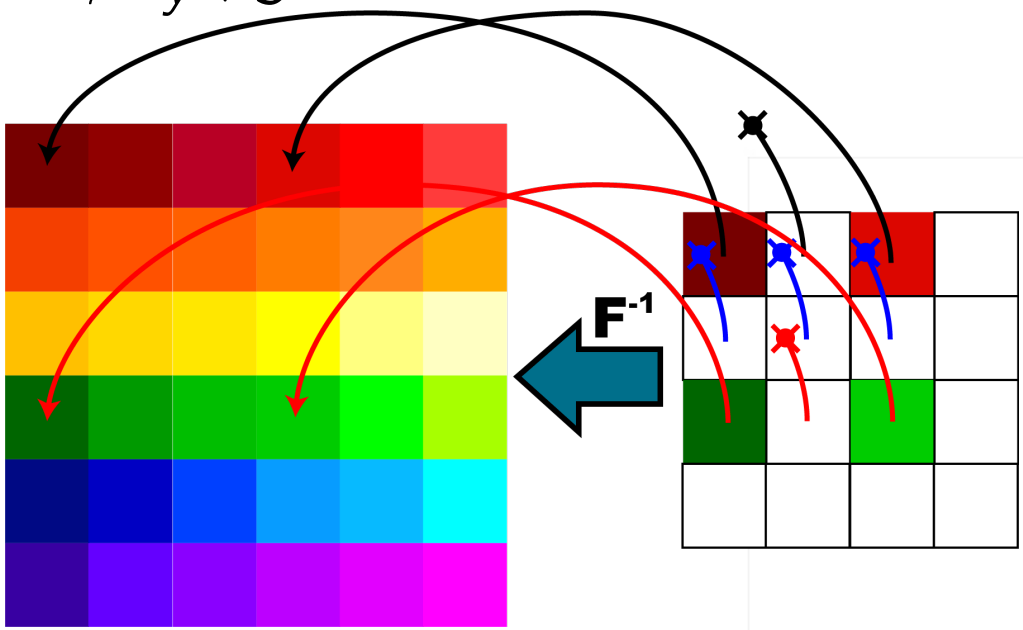


Whatever the transformation F, we **always use** the strategy which consists for each pixel of the **final image** to find the corresponding pixel on the **initial image** according to $\mathbf{F}^{-1}$

Resize: how to ? (general case)

If the **final size** is **not a divisor** of the initial one, then the **coordinates** of the corresponding pixel calculated with the **transformation** F^{-1} are **not integers**!

$$F^{-1}(x,y): \begin{cases} x & \mapsto & x*S \\ y & \mapsto & y*S \end{cases} \quad \forall S \mid (width, height) \not\equiv 0[S],\ \forall x \in \mathbb{N},\ \forall y \in \mathbb{N}$$

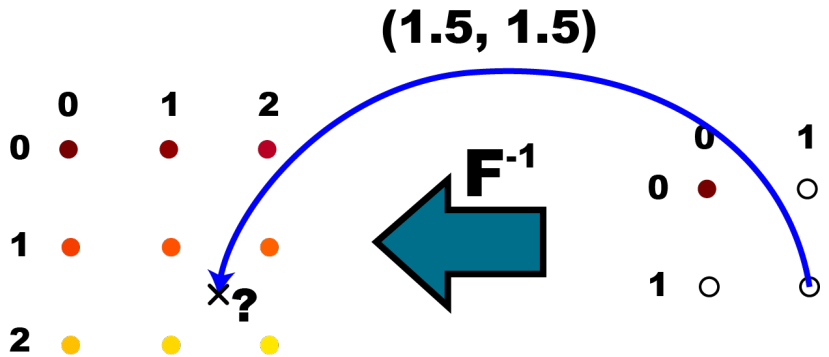


[L. Avanthey et al., 2018]

How shall we do it then?

Resize, general case : interpolations

Since the **projection** of the pixels of the final image **does not correspond exactly** to the pixels of the initial image (ie. the projected pixels do not have **integer coordinates**), the information sought (pixel value) must be **created from known values** (pixels around): this is called **data interpolation**



[L. Avanthey et al., 2018]

Different interpolation functions can be used depending on the **number of known pixels** involved in the modeling

Interpolations

What **kind of functions** can we use as a **model**?

Interpolations

What **kind of functions** can we use as a **model**?

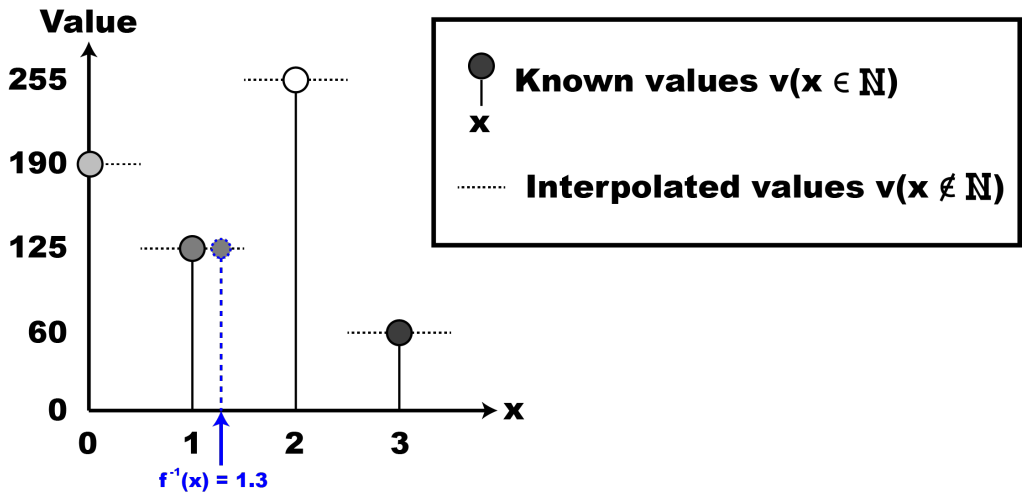
Polynomial functions!

Let's start with the simplest case: degree 0

0-D Polynomial: Nearest Neighbor interpolation – 1D case

The **Nearest-Neighbor interpolation** consists of setting the **final value** to the **value of the nearest known point** of the projected position

- Let's **forget the pixels** and first see the **1D case**
  - ⇒ The 1D model used is a 0 degree polynomial (i.e. a constant)



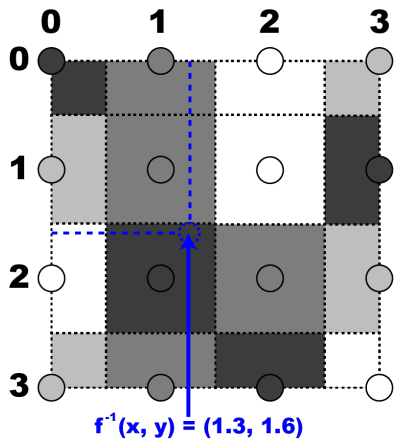
[cc by sa L. Avanthey et al., 2018]

- Let F be any transformation,
  for $x_1 \in \mathbb{N}$,
  then $F^{-1}(x_1) = 1.3 \notin \mathbb{N}$
- With an interpolation to nearest neighbor we have:

$$F^{-1}(x_1) \Leftrightarrow 1$$
and thus:
$$v(F^{-1}(x_1)) \Leftrightarrow v(1) = 125$$

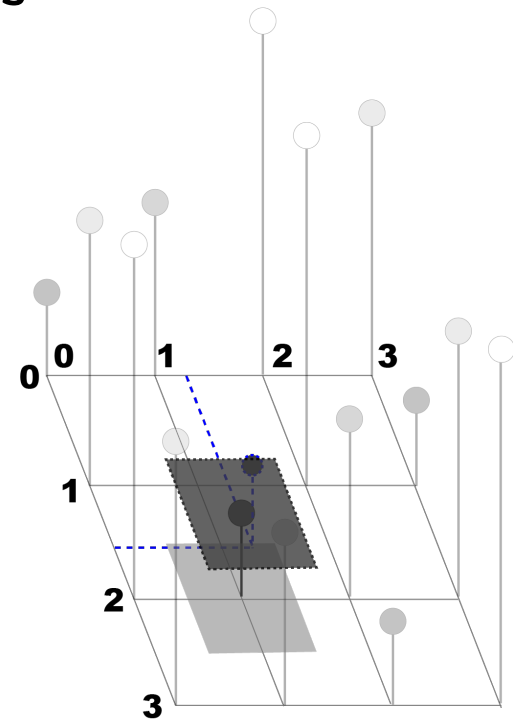
Nearest Neighbor interpolation – 2D case

- Let's see now the **2D case** with our **pixels**



x
y ● **Known values v(x ∈ ℕ, y ∈ ℕ)**

Interpolated values **v(x ∉ ℕ, y ∉ ℕ)**
v(x ∉ ℕ, y ∈ ℕ)
v(x ∈ ℕ, y ∉ ℕ)



[L. Avanthey et al., 2018]

- Let F be any transformation, for $x_1 \in \mathbb{N}$ and $y_1 \in \mathbb{N}$, then $F^{-1}(x_1, y_1) = (1.3, 1.6) \notin \mathbb{N}$
- With an interpolation to nearest neighbor we have:
  $F^{-1}(x_1, y_1) \Leftrightarrow (1, 2)$
  and thus: $v(F^{-1}(x_1, y_1)) \Leftrightarrow v(1, 2) = 60$

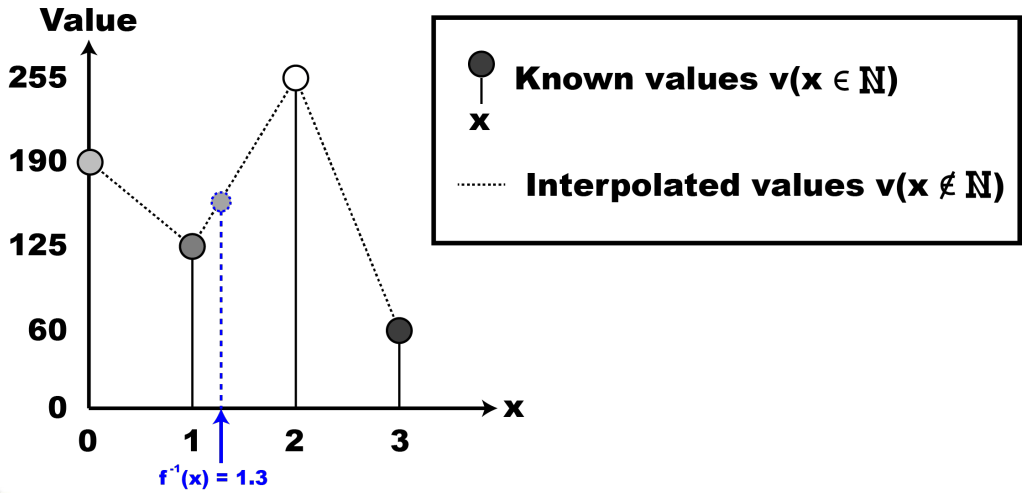
Next?

What else can we try?

1-D polynomial: Linear interpolation – 1D Case

- Again, let's **forget the pixels** and first see the more simple **1D case**

The **linear interpolation** uses **2 known values** and a **1 degree polynomial** model: $v(x) = ax + b$



[cc L. Avanthey et al., 2018]

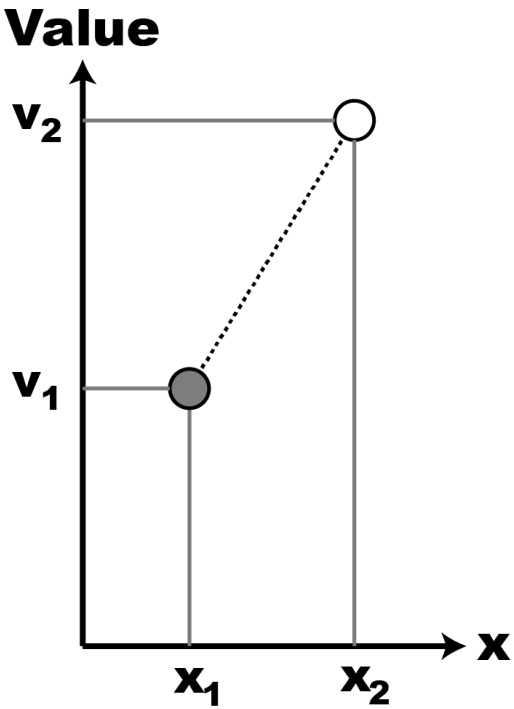
- Let F be any transformation, $x_1 \in \mathbb{N}$, $F^{-1}(x_1) = 1.3 \notin \mathbb{N}$ and $v(x) = 130x - 5 \ \forall x \in [1, 2]$
- With a linear interpolation, we have:

$$v(F^{-1}(x_1)) = 130 * F^{-1}(x_1) - 5$$
and thus:
$$v(F^{-1}(x_1)) = 164$$

? $a = 130$ and $b = -5$?

Linear interpolation: how to calculate the coefficients

$$\begin{cases} v_1 = ax_1 + b \\ v_2 = ax_2 + b \end{cases}$$

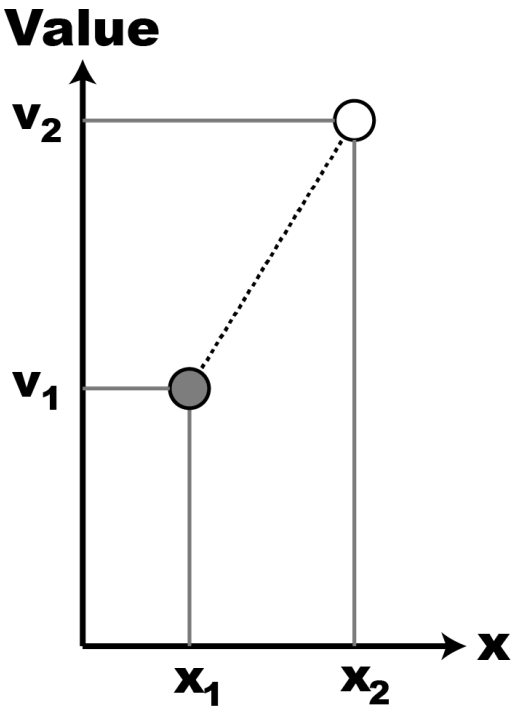


[L. Avanthey et al., 2018]

Linear interpolation: how to calculate the coefficients

$$\begin{cases} v_1 = ax_1 + b \\ v_2 = ax_2 + b \end{cases}$$

$$\Leftrightarrow \begin{cases} v_1 = ax_1 + b \\ b = v_2 - ax_2 \end{cases}$$



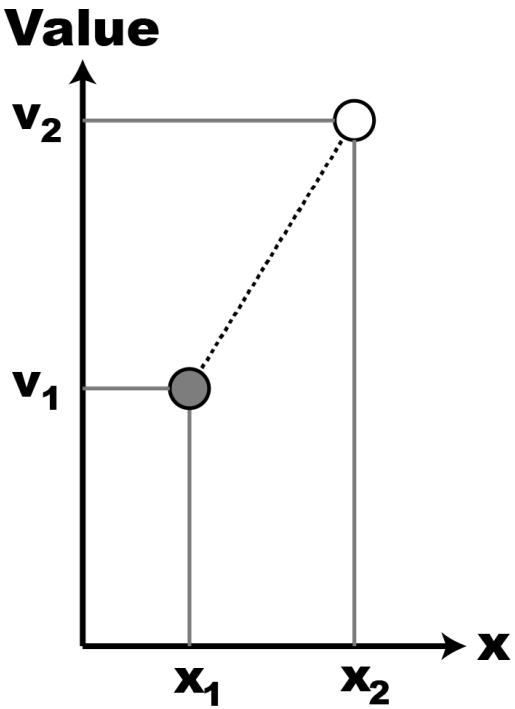
[CC BY L. Avanthey et al., 2018]

Linear interpolation: how to calculate the coefficients

$$\begin{cases} v_1 = ax_1 + b \\ v_2 = ax_2 + b \end{cases}$$

$$\Leftrightarrow \begin{cases} v_1 = ax_1 + b \\ b = v_2 - ax_2 \end{cases}$$

$$\Leftrightarrow \begin{cases} v_1 = ax_1 + (v_2 - ax_2) \\ b = v_2 - ax_2 \end{cases}$$



[CC BY NC SA L. Avanthey et al., 2018]

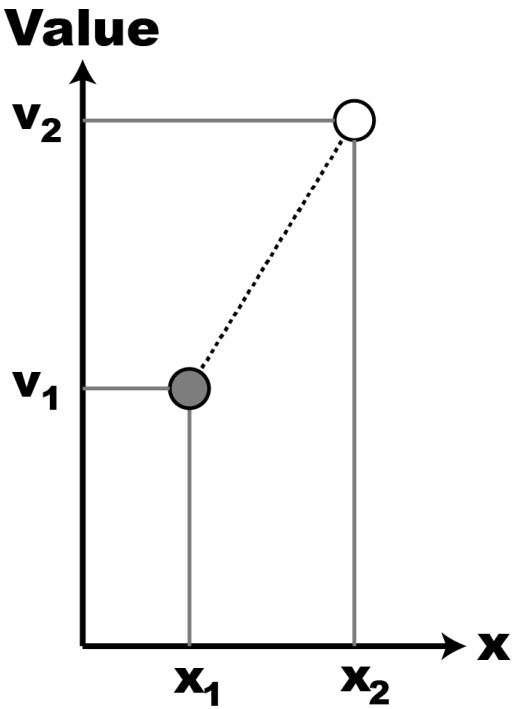
Linear interpolation: how to calculate the coefficients

$$\begin{cases} v_1 = ax_1 + b \\ v_2 = ax_2 + b \end{cases}$$

$$\Leftrightarrow \begin{cases} v_1 = ax_1 + b \\ b = v_2 - ax_2 \end{cases}$$

$$\Leftrightarrow \begin{cases} v_1 = ax_1 + (v_2 - ax_2) \\ b = v_2 - ax_2 \end{cases}$$

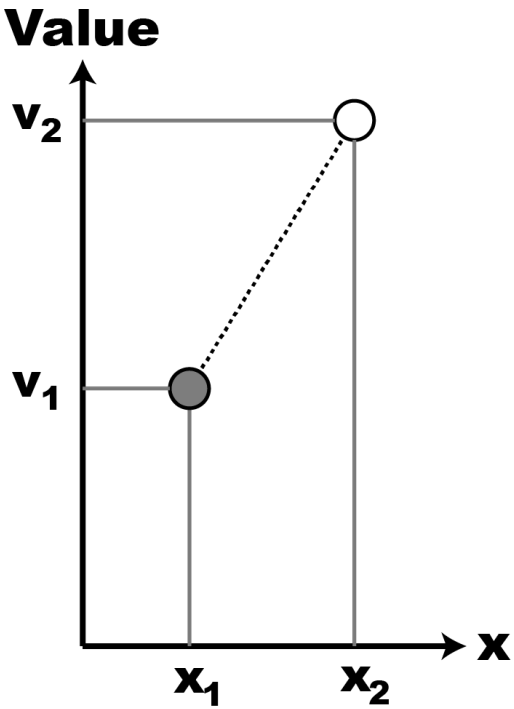
$$\Leftrightarrow \begin{cases} a = \frac{v_2 - v_1}{x_2 - x_1} \\ b = v_2 - \left(\frac{v_2 - v_1}{x_2 - x_1}\right) x_2 \end{cases}$$



[CC BY NC SA L. Avanthey et al., 2018]

Linear interpolation: how to calculate the coefficients

$$\begin{cases} v_1 = ax_1 + b \\ v_2 = ax_2 + b \end{cases}$$



[L. Avanthey et al., 2018]

$$\Leftrightarrow \begin{cases} v_1 = ax_1 + b \\ b = v_2 - ax_2 \end{cases}$$

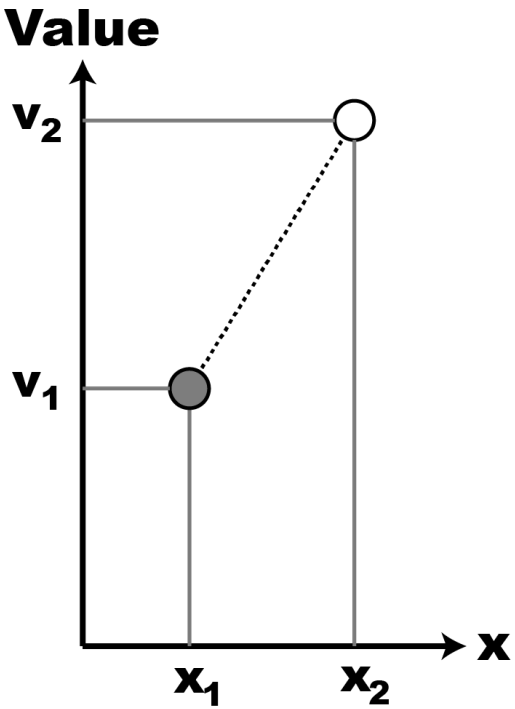
$$\Leftrightarrow \begin{cases} v_1 = ax_1 + (v_2 - ax_2) \\ b = v_2 - ax_2 \end{cases}$$

$$\Leftrightarrow \begin{cases} a = \frac{v_2 - v_1}{x_2 - x_1} \\ b = v_2 - \left(\frac{v_2 - v_1}{x_2 - x_1}\right) x_2 \end{cases}$$

$$\Leftrightarrow \begin{cases} a = \frac{v_2 - v_1}{x_2 - x_1} \\ b = \frac{v_1 x_2 - v_2 x_1}{x_2 - x_1} \end{cases}$$

Linear interpolation: how to calculate the coefficients

$$\begin{cases} v_1 = ax_1 + b \\ v_2 = ax_2 + b \end{cases}$$



[L. Avanthey et al., 2018]

$$\Leftrightarrow \begin{cases} v_1 = ax_1 + b \\ b = v_2 - ax_2 \end{cases}$$

$$\Leftrightarrow \begin{cases} v_1 = ax_1 + (v_2 - ax_2) \\ b = v_2 - ax_2 \end{cases}$$

$$\Leftrightarrow \begin{cases} a = \frac{v_2 - v_1}{x_2 - x_1} \\ b = v_2 - \left(\frac{v_2 - v_1}{x_2 - x_1}\right) x_2 \end{cases}$$

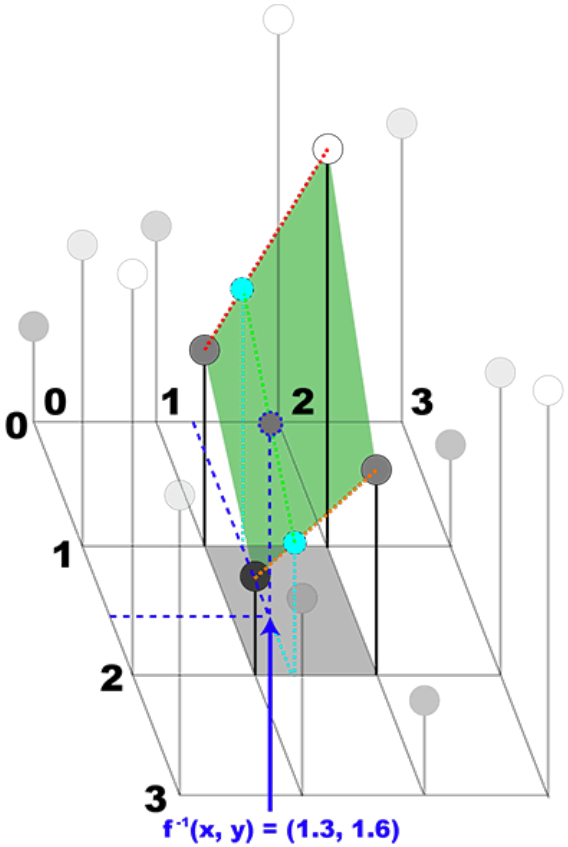
$$\Leftrightarrow \begin{cases} a = \frac{v_2 - v_1}{x_2 - x_1} \\ b = \frac{v_1 x_2 - v_2 x_1}{x_2 - x_1} \end{cases}$$

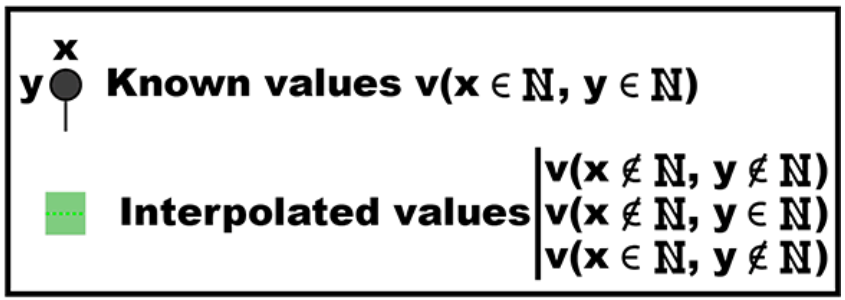
$x2 - x1 = 1$, so:

$$\Leftrightarrow \begin{cases} a = v_2 - v_1 \\ b = v_1 x_2 - v_2 x_1 \end{cases}$$

Bilinear interpolation – 2D case

The **bilinear interpolation** (2D) can be divided into **3 linear interpolations** (1D): **2** along the **x-axis** (above and below), and the **3rd** one along the **y-axis**

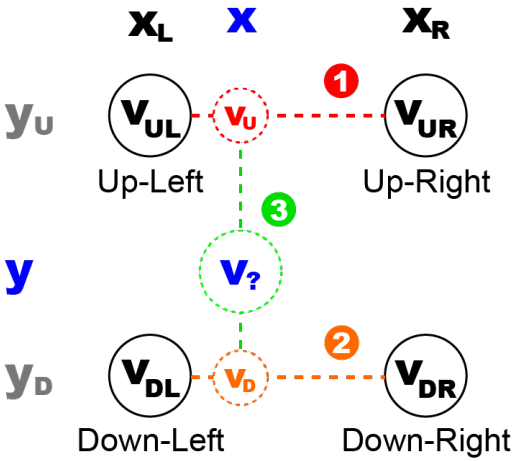




[L. Avanthey et al., 2018]

Bilinear interpolation: how to?

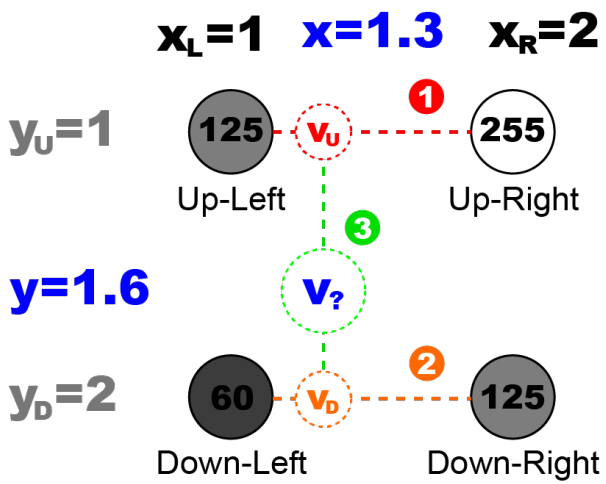
- **We search for** $v(x, y)$ $\forall x \notin \mathbb{N}$ and/or $y \notin \mathbb{N}$



[CC BY L. Avanthey et al., 2018]

1. Interpolation along the **x-axis**: **up**
   - compute a_u and b_u using $(x_L,\ v_{UL})$ and $(x_R,\ v_{UR})$
   - compute the upside interpolated value of x: $v_u(x) = a_u x + b_u$
2. Interpolation along **x-axis**: **down**
   - compute a_d and b_d using $(x_L,\ v_{DL})$ and $(x_R,\ v_{DR})$
   - compute the downside interpolated value of x: $v_d(x) = a_d x + b_d$
3. Interpolation along **y-axis**: **final**
   - compute a and b using $(y_u,\ v_u)$ and $(y_d,\ v_d)$
   - compute the interpolated value of y: $v(y) = ay + b \Leftrightarrow v(x, y)$

Bilinear interpolation example



[© L. Avanthey et al., 2018]

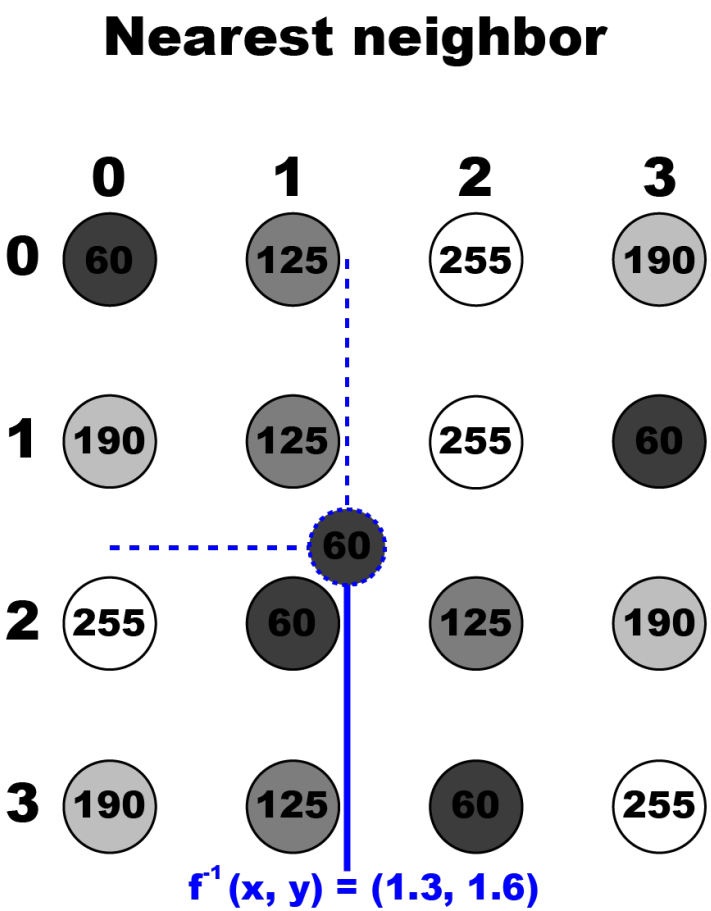
1. Along **x-axis**: **up**
   - $a_u = 255 - 125 = 130$
     $b_u = 125 \times 2 - 255 \times 1 = -5$
   - $v_u(1.3) = a_u \times 1.3 + b_u = 164$
2. Along **x-axis**: **down**
   - $a_d = 125 - 60 = 65$
     $b_d = 60 \times 2 - 125 \times 1 = -5$
   - $v_d(1.3) = a_d \times 1.3 + b_d = 79.5$
3. Along **y-axis**: **final**
   - $a = 79.5 - 164 = -84.5$
     $b = 164 \times 2 - 79.5 \times 1 = 248.5$
   - $v(1.6) = a \times 1.6 + b = 113.3$

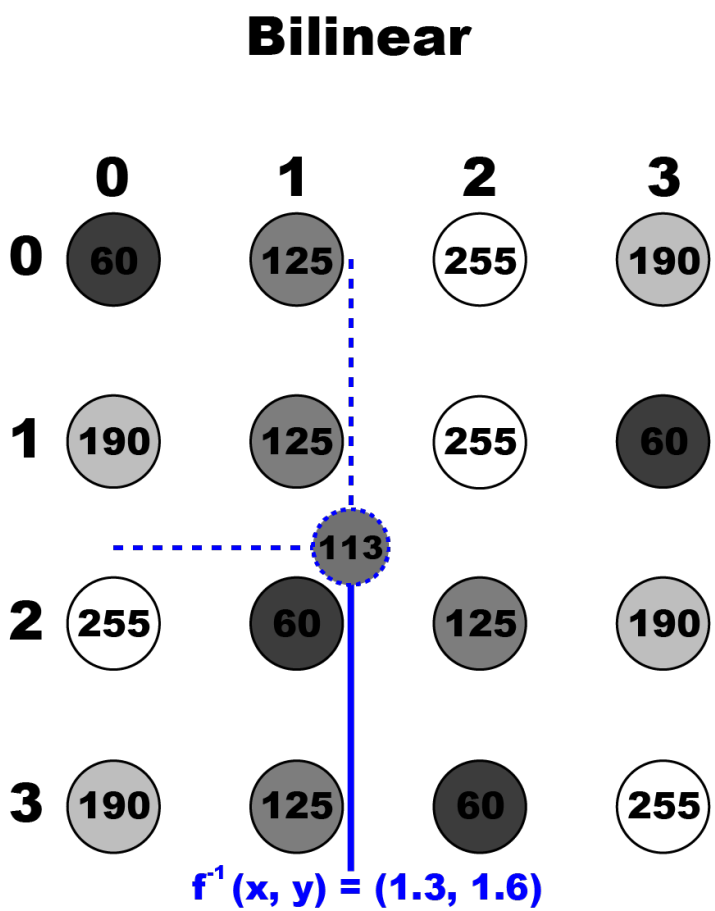
$\Rightarrow v(1.3, 1.6) = 113.3$

Bilinear interpolation: symmetrical?

If we **first** interpolate on the **y-axis** twice (left, right) and **then** on the **x-axis**, do we obtain the **same results** ?

2D Nearest Neighbor interpolation vs Bilinear interpolation





[L. Avanthey et al., 2018]

Next polynomial?

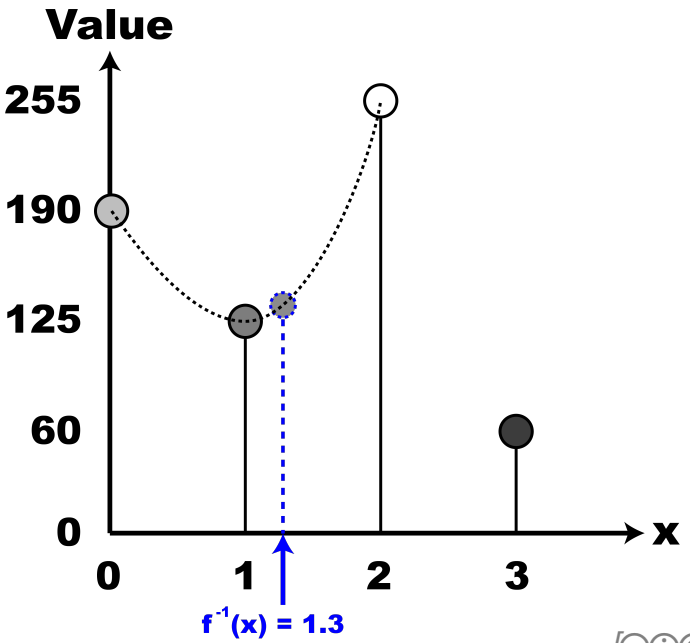
To get a **smoother** estimate, what is the **next degree** to use for our polynomial?

Degree 2?

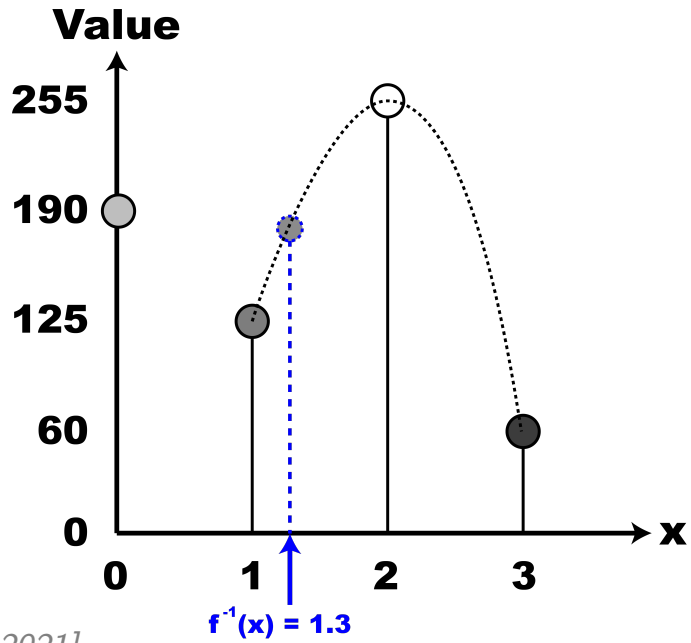
$\Rightarrow$ Let's take a **degree 2** polynomial ($v(x) = ax^2 + bx + c$): there are **3 unknowns**, so we need **3 points** to determine its equation.

Degree 2?

$\Rightarrow$ Let's take a **degree 2** polynomial ($v(x) = ax^2 + bx + c$): there are **3 unknowns**, so we need **3 points** to determine its equation.



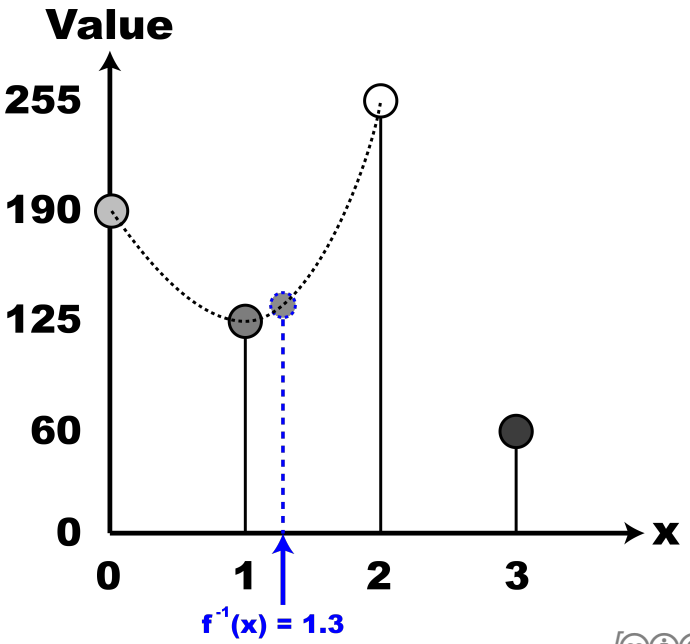
OR



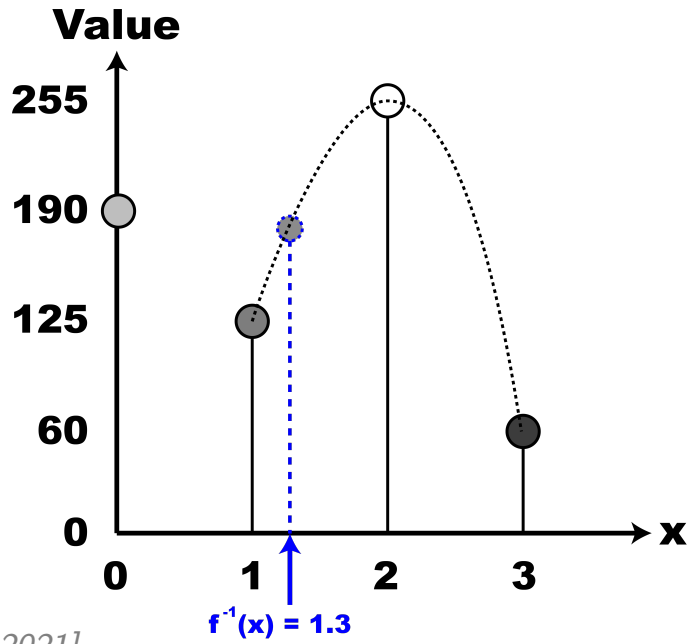
[L. Avanthey et al., 2021]

Degree 2?

$\Rightarrow$ Let's take a **degree 2** polynomial ($v(x) = ax^2 + bx + c$): there are **3 unknowns**, so we need **3 points** to determine its equation.



OR



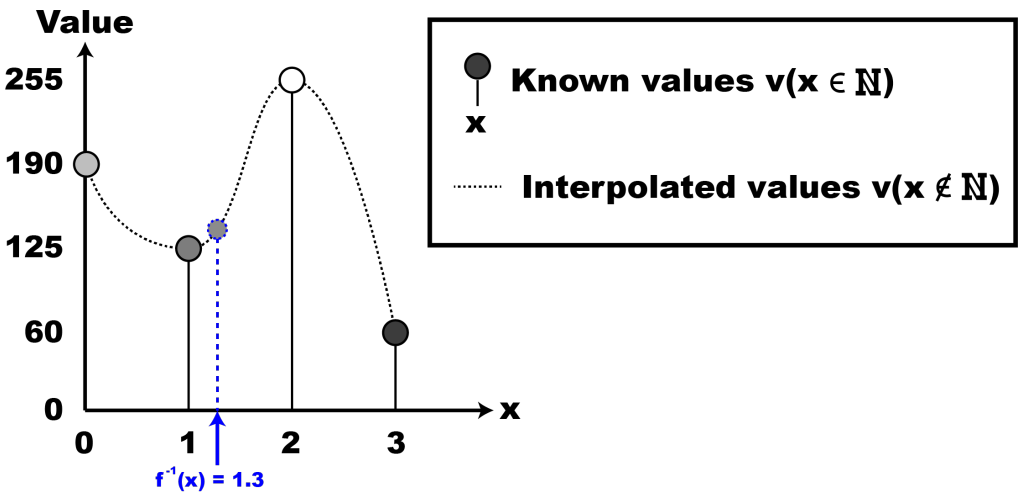
[L. Avanthey et al., 2021]

So we will prefer to use a polynomial of **degree 3** to have a **symmetry** and not to privilege a direction!

3-D polynomial : Cubic interpolation – 1D Case

- Once again, let's **forget the pixels** and first see the more simple **1D case**

The **cubic interpolation** uses **4 known values** and a **3 degree polynomial** model: $v(x) = ax^3 + bx^2 + cx + d$

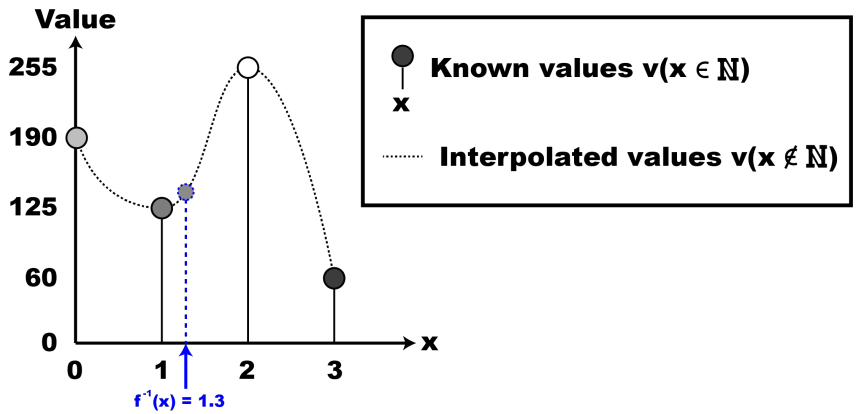


[L. Avanthey et al., 2018]

- Let F be any transformation, $\forall x_1 \in \mathbb{N}$, and $F^{-1}(x_1) \notin \mathbb{N}$
- We search for **4 unknowns**: a, b, c and d, so **4 known values are needed** to calculate them
- The **4 known values** are taken **symmetrically**: 2 before and 2 after the interpolated value

Linear interpolation: how to calculate the coefficients

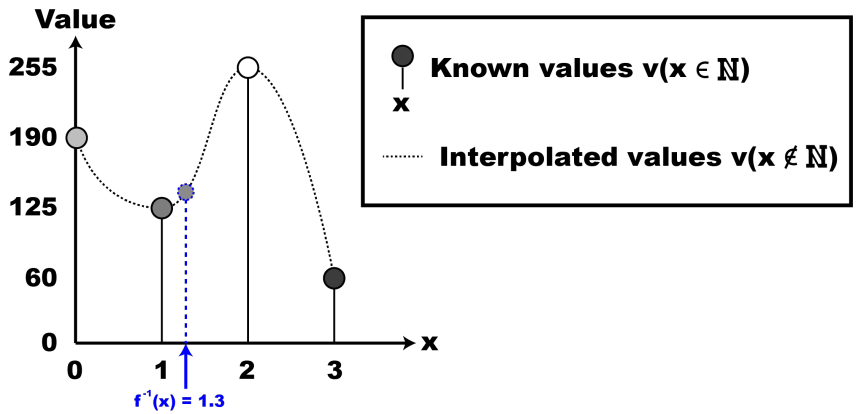
$$\begin{cases} 0+0+0+d=190 \\ a+b+c+d=125 \\ 8a+4b+2c+d=255 \\ 27a+9b+3c+d=60 \end{cases}$$



[CC BY L. Avanthey et al., 2018]

Linear interpolation: how to calculate the coefficients

$$\begin{cases} 0+0+0+d=190 \\ a+b+c+d=125 \\ 8a+4b+2c+d=255 \\ 27a+9b+3c+d=60 \end{cases} \Leftrightarrow \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 \\ 27 & 9 & 3 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 190 \\ 125 \\ 255 \\ 60 \end{pmatrix}$$



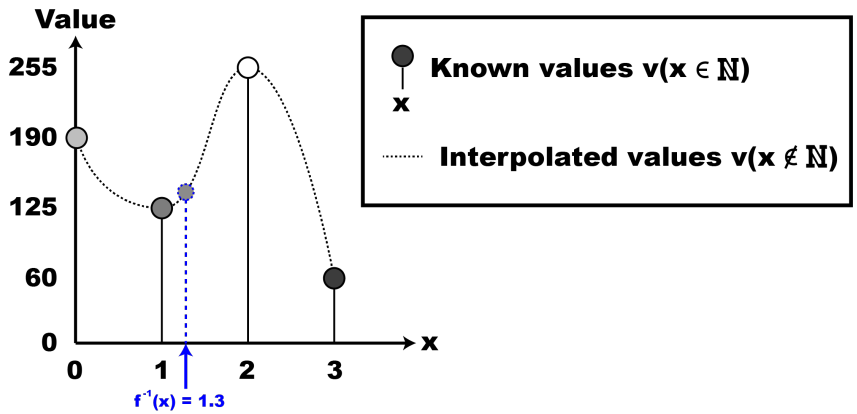
[CC BY L. Avanthey et al., 2018]

Linear interpolation: how to calculate the coefficients

$$\begin{cases} 0+0+0+d=190 \\ a+b+c+d=125 \\ 8a+4b+2c+d=255 \\ 27a+9b+3c+d=60 \end{cases} \Leftrightarrow \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 \\ 27 & 9 & 3 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 190 \\ 125 \\ 255 \\ 60 \end{pmatrix}$$

Matrix inversion (`.inv()` method in `OpenCV`)

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} -0.2 & 0.5 & -0.5 & 0.2 \\ 1 & -2.5 & 2 & -0.5 \\ -1.8 & 3 & -1.5 & 0.3 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 190 \\ 125 \\ 255 \\ 60 \end{pmatrix}$$



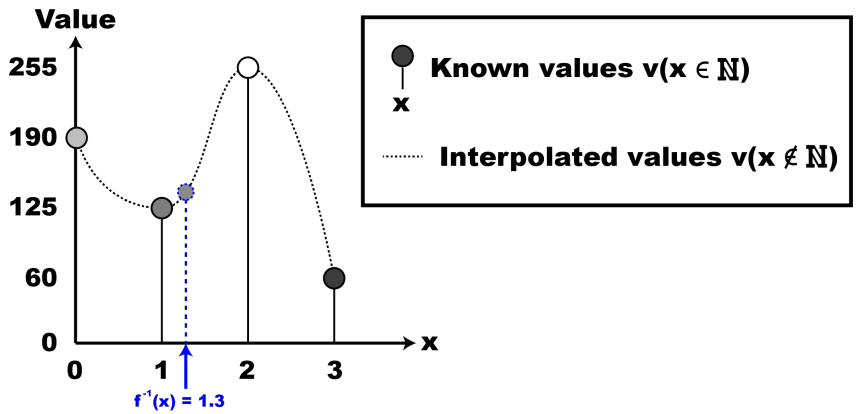
[L. Avanthey et al., 2018]

Linear interpolation: how to calculate the coefficients

$$\begin{cases} 0+0+0+d=190 \\ a+b+c+d=125 \\ 8a+4b+2c+d=255 \\ 27a+9b+3c+d=60 \end{cases} \Leftrightarrow \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 \\ 27 & 9 & 3 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 190 \\ 125 \\ 255 \\ 60 \end{pmatrix}$$

Matrix inversion (`.inv()` method in `OpenCV`)

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} -0.2 & 0.5 & -0.5 & 0.2 \\ 1 & -2.5 & 2 & -0.5 \\ -1.8 & 3 & -1.5 & 0.3 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 190 \\ 125 \\ 255 \\ 60 \end{pmatrix}$$

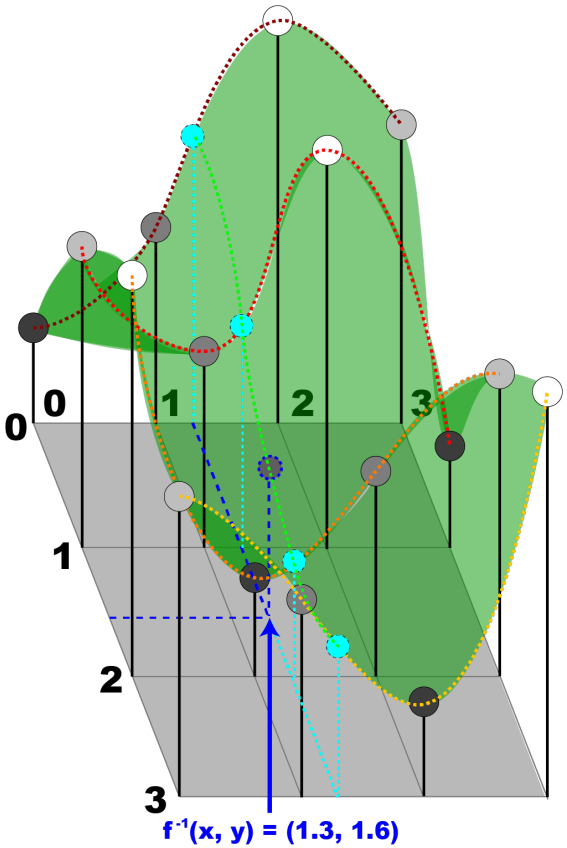


[ⓒⓘⓢ L. Avanthey et al., 2018]

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} -86.7 \\ 357 \\ -335.8 \\ 190 \end{pmatrix}$$

Bicubic interpolation – 2D Case

The **2D bicubic interpolation** can be divided into **5 1D cubic interpolations**: **4** along the **x-axis**, and the **5th** one along the **y-axis**



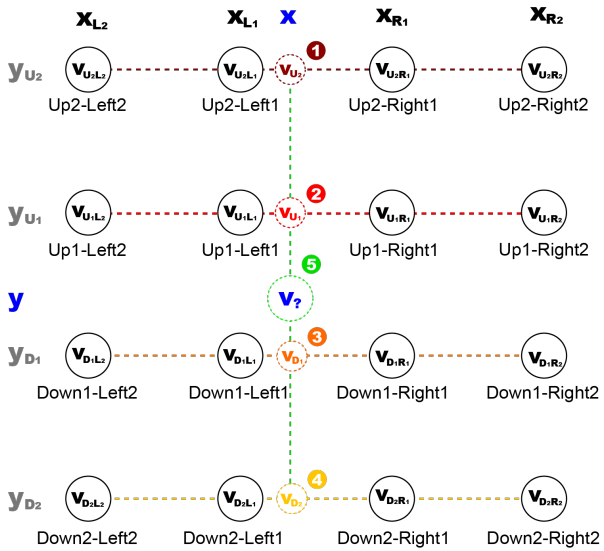
x
y Known values v(x ∈ ℕ, y ∈ ℕ)

Interpolated values
v(x ∉ ℕ, y ∉ ℕ)
v(x ∉ ℕ, y ∈ ℕ)
v(x ∈ ℕ, y ∉ ℕ)

[L. Avanthey et al., 2018]

Bicubic interpolation: how to?

- **We search for** $v(x, y)$ $\forall x \notin \mathbb{N}$ and/or $y \notin \mathbb{N}$

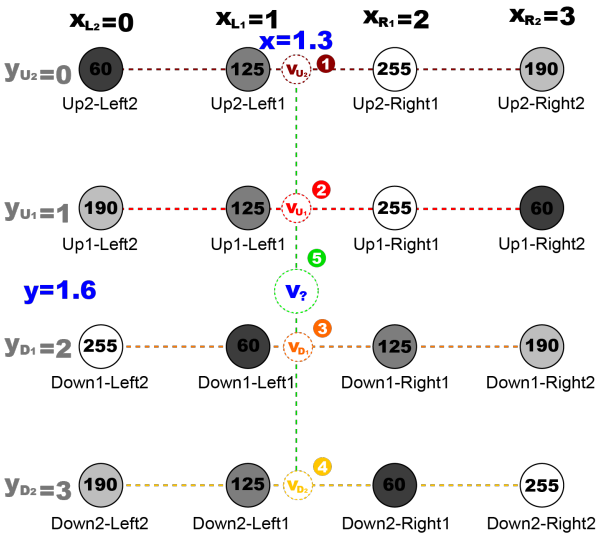


[L. Avanthey et al., 2018]

1. Interpolation along the **x-axis**: **up 2**
   - $v_{u_2}(x) = a_{u_2}x^3 + b_{u_2}x^2 + c_{u_2}x + d_{u_2}$
2. Interpolation along the **x-axis**: **up 1**
   - $v_{u_1}(x) = a_{u_1}x^3 + b_{u_1}x^2 + c_{u_1}x + d_{u_1}$
3. Interpolation along **x-axis**: **down 1**
   - $v_{d_1}(x) = a_{d_1}x^3 + b_{d_1}x^2 + c_{d_1}x + d_{d_1}$
4. Interpolation along **x-axis**: **down 2**
   - $v_{d_2}(x) = a_{d_2}x^3 + b_{d_2}x^2 + c_{d_2}x + d_{d_2}$
5. Interpolation along **y-axis**: **final**
   - $v(y) = ay^3 + by^2 + cy + d \Leftrightarrow v(x, y)$

Bicubic interpolation example

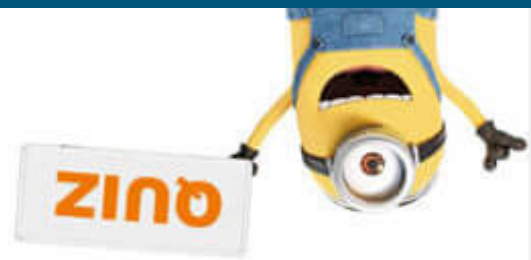
$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} -0.2 & 0.5 & -0.5 & 0.2 \\ 1 & -2.5 & 2 & -0.5 \\ -1.8 & 3 & -1.5 & 0.3 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{pmatrix}$$



[L. Avanthey et al., 2020]

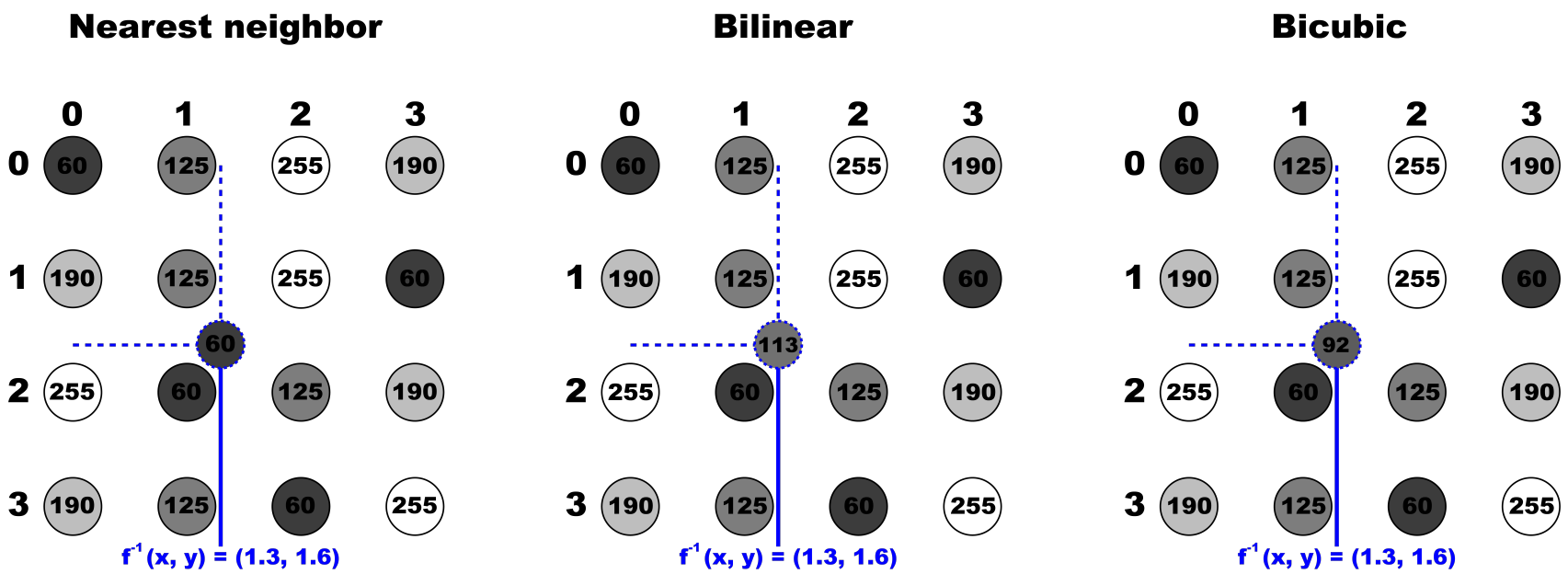
1. Interpolation along the **x-axis**: **up 2**
   - $v_{u_2}(x) = -39 \times 1.3^3 + 162.5 \times 1.3^2 - 58.5 \times 1.3 + 60 = 172.892$
2. Interpolation along the **x-axis**: **up 1**
   - $v_{u_1}(x) = -91 \times 1.3^3 + 357.5 \times 1.3^2 - 331.5 \times 1.3 + 190 = 163.298$
3. Interpolation along **x-axis**: **down 1**
   - $v_{d_1}(x) = -45.5 \times 1.3^3 + 260 \times 1.3^2 - 409.5 \times 1.3 + 255 = 62.0865$
4. Interpolation along **x-axis**: **down 2**
   - $v_{d_2}(x) = 45.5 \times 1.3^3 - 130 \times 1.3^2 + 19.5 \times 1.3 + 190 = 95.6135$
4. Interpolation along **y-axis**: **final**
   - $v(y) = 35.2 \times 1.6^3 - 159.0 \times 1.6^2 + 114.2 \times 1.6 + 173.0 = 92.649 \Leftrightarrow v(x, y)$

Bicubic interpolation: symmetrical?



If we **first** interpolate on the **y-axis** 4 times (left 2, left 1, right 1, right 2) and **then** on the **x-axis**, do we obtain the **same results** ?

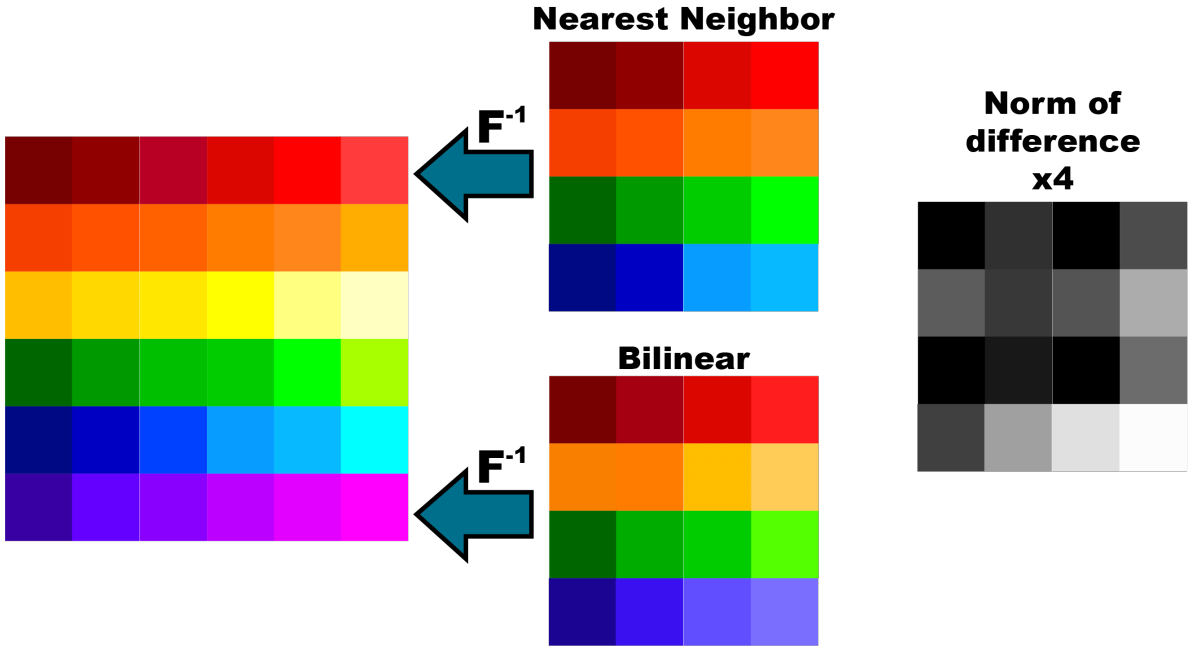
2D Nearest Neighbor vs Bilinear vs Bicubic interpolation



[L. Avanthey et al., 2021]

Back to the general case of resize

We can now **calculate** a value by **interpolation** for **each pixel** of our **final image**

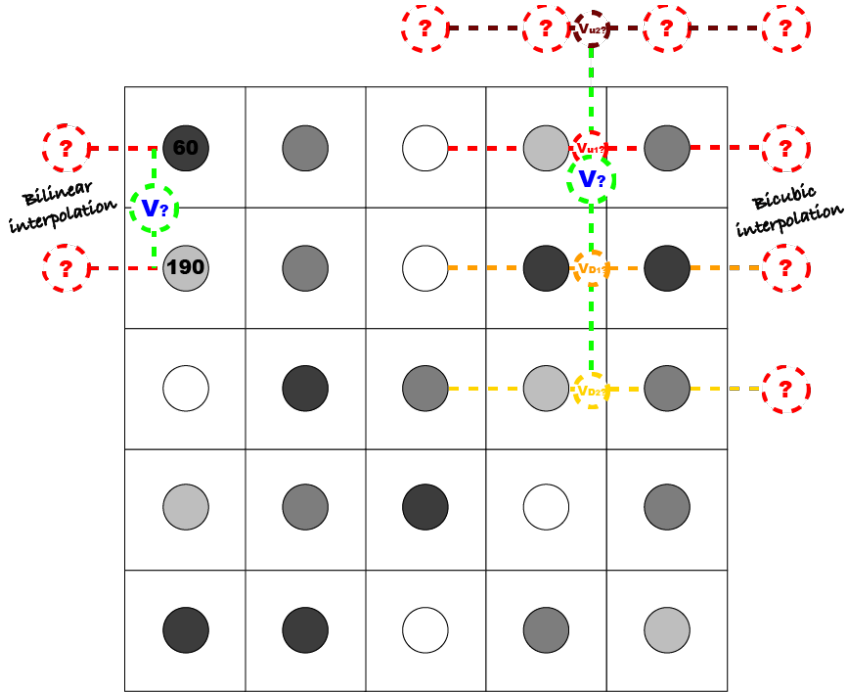


●●●

[CC BY L. Avanthey et al., 2018]

The **results** will obviously be **different** depending on the **type of interpolation** used, but the **overall result** remains very **similar** to the **eye**

What to do with the edges?



?

[CC BY L. Avanthey et al., 2020]

We **can not** calculate an **interpolation** other than the nearest neighbor interpolation at the **edges**

Some ways to manage the edges

There are **plenty of ways** to manage the **edges**

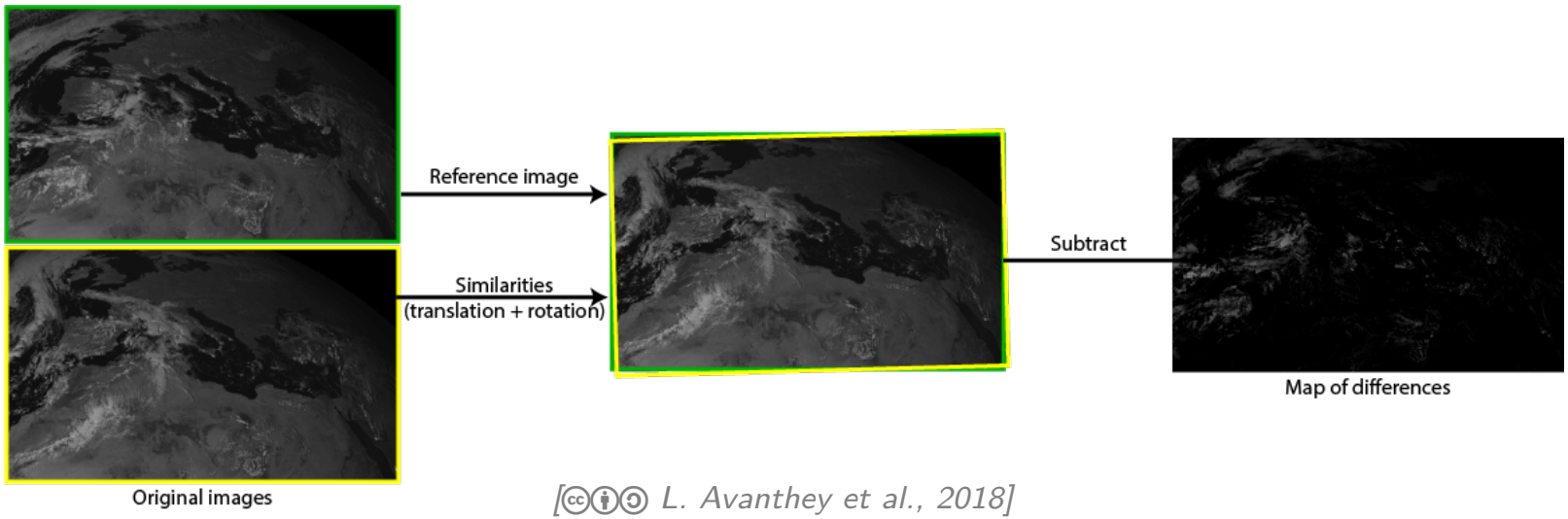
In our applications, we generally prefer **not to treat them** (edges to 0), but we can also sometimes use the **mirror method** or the **replication method**

[L. Beaudoin et al., 2018]

Similarities: registrations (simple case) (1)

To **detect changes** on temporal series, to **superimpose** images on maps or on other images (multi-spectral/scale) or to create **mosaics**, we needs to **register** the images in the **same frame**

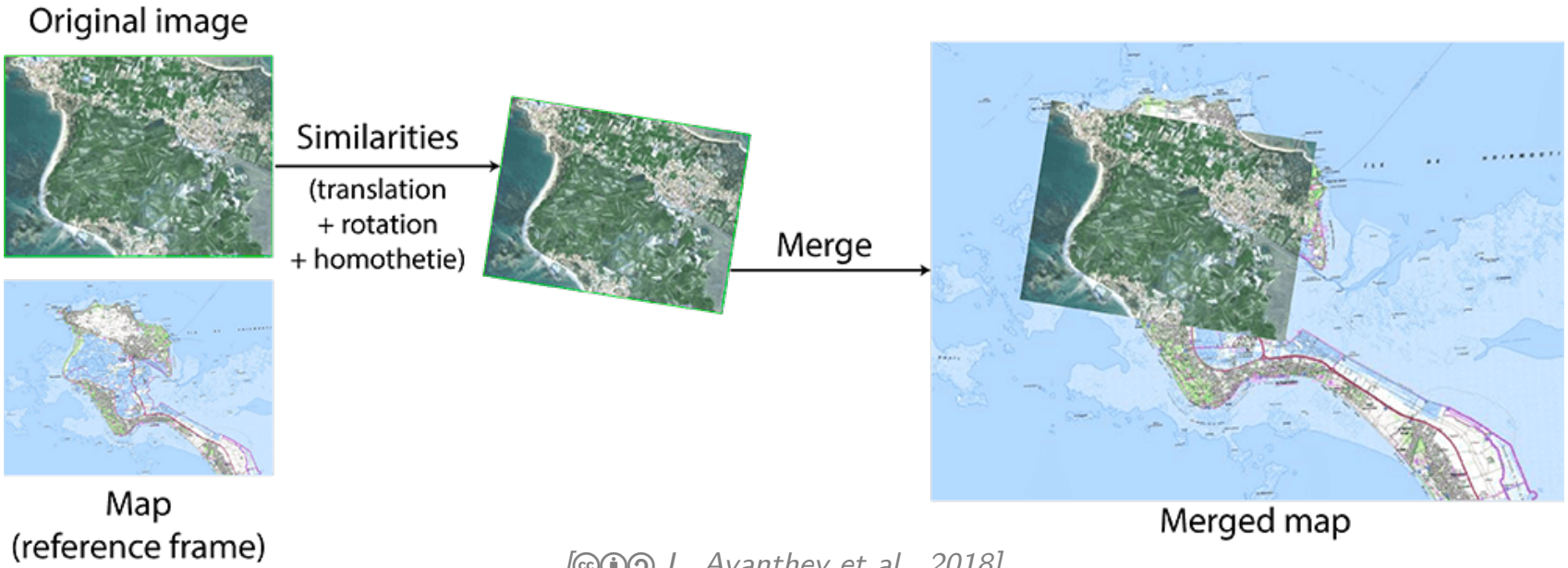
When the **parallelism** of the objects observed is **preserved**, we can then use **similarities** for the registration



[L. Avanthey et al., 2018]

Similarities: registrations (simple case) (2)

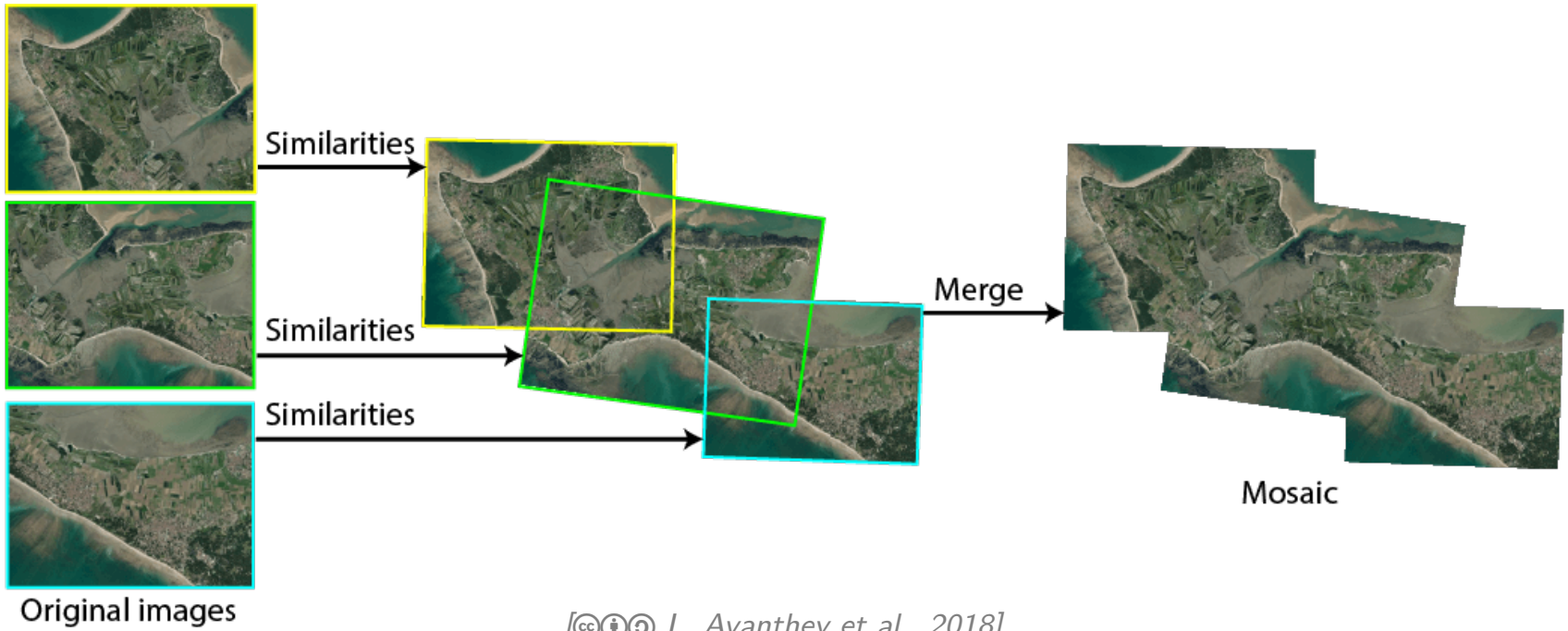
Preservation of the **parallelism** of objects means that the **angle** between the **normal** of the **scene plane** and that of the **image plane** is the **same** for the **whole** series



[L. Avanthey et al., 2018]

Similarities: registrations (simple case) (3)

When all the normals of a series of images are parallel to each other, we speak of **fronto-parallel shots**

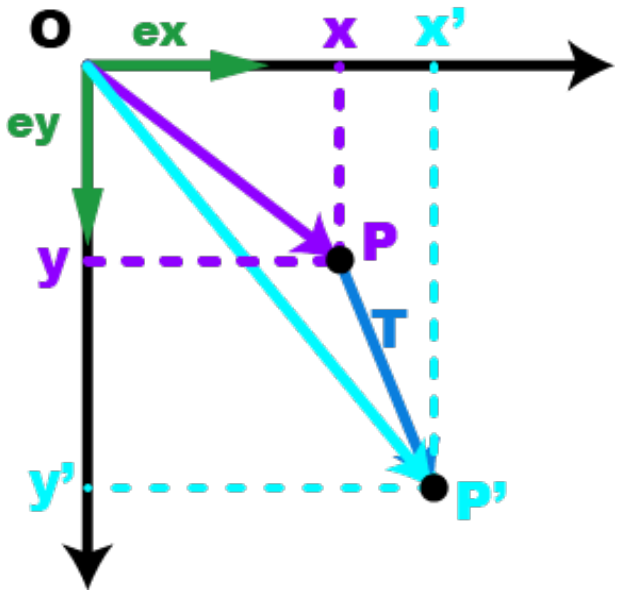


[L. Avanthey et al., 2018]

A similarity is a combination of a translation (T), a scale factor (s) and a rotation (R)

Similarities: how to ? – Translations

If we apply a **translation transformation**, then the **coordinates** P' result from the **sum** of T (the **translation** between P and P') and the **coordinates** P: $P' = P + T$



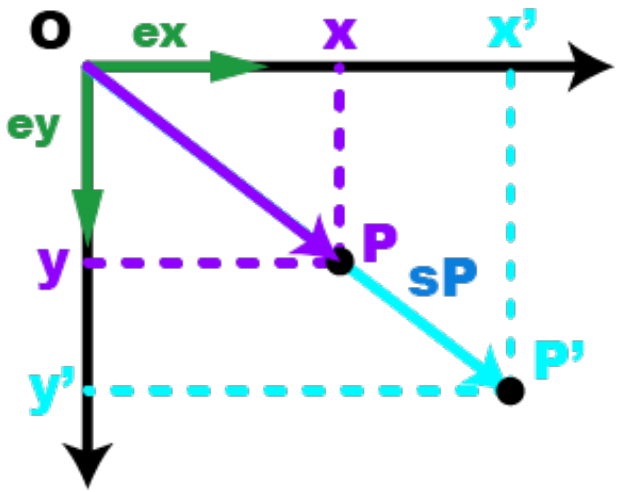
[cc L. Avanthey et al., 2018]

Euclidian form:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} T_x \\ T_y \end{pmatrix}$$

Similarities: how to ? – Homotheties

If we apply a **homothetie transformation**, then the **coordinates** P' result from the **multiplication** of the scalar s (the **scale factor** between P and P') and the **coordinates** P: $P' = sP$



[CC BY NC SA L. Avanthey et al., 2020]

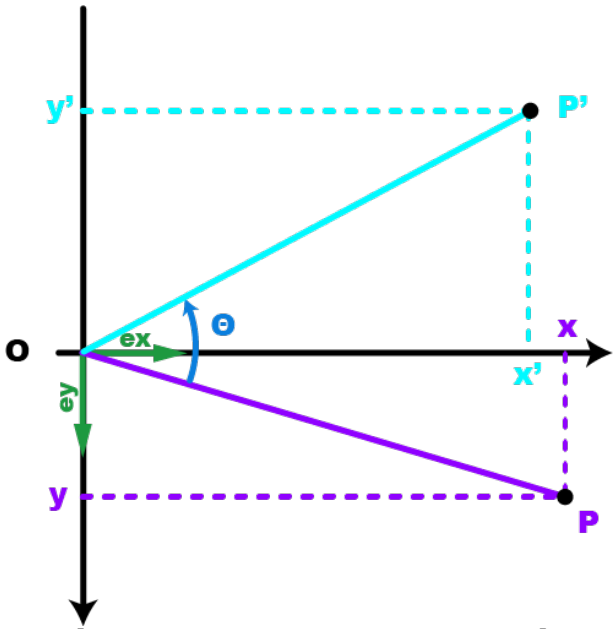
Euclidian form:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s \times x \\ s \times y \end{pmatrix}$$

Similarities: how to ? – Central rotations (simple case) (1)

If we apply a **central rotation transformation**, the **coordinates** P' result from the **multiplication** of the transition matrix R (the **rotation** of center O between P and P') with the **coordinates** P:

$$P' = RP$$



[CC BY L. Avanthey et al., 2018]

It is **not easy** to find R, the **relationship** between the **coordinates** of P and P', but we know that:

$$\underbrace{\begin{pmatrix} x' \\ y' \end{pmatrix}}_{P'} = \underbrace{\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}}_{R} \underbrace{\begin{pmatrix} x \\ y \end{pmatrix}}_{P}$$

$$\text{and } \begin{pmatrix} x \\ y \end{pmatrix} = (x\vec{e_x} + y\vec{e_y})$$

Similarities: how to ? – Central rotations (simple case) (2)

$$\begin{aligned} P' &= R(x\vec{e_x} + y\vec{e_y}) \\ &\Leftrightarrow xR\vec{e_x} + yR\vec{e_y} \end{aligned}$$

- Let $R\vec{e_x} = \vec{e_x}'$ and $R\vec{e_y} = \vec{e_y}'$

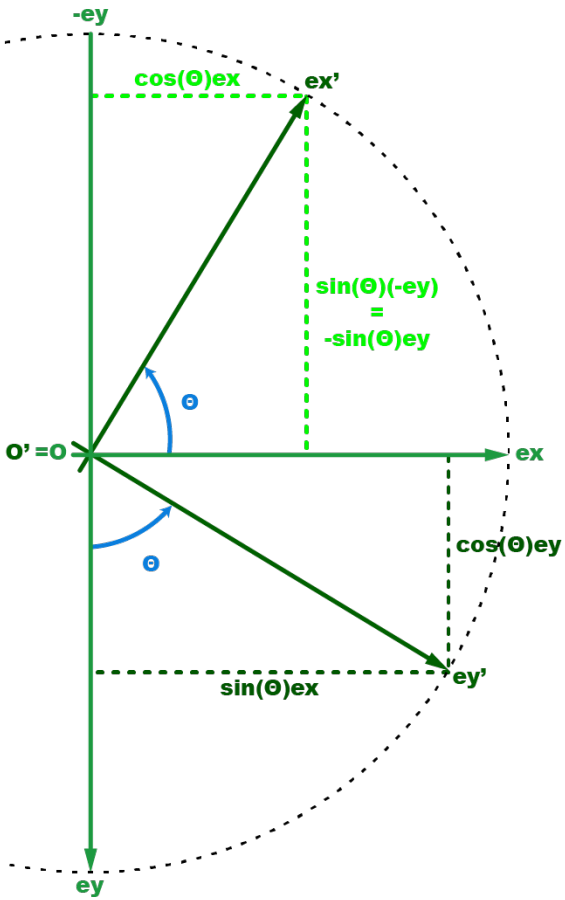
$$\Rightarrow P' = x\vec{e_x}' + y\vec{e_y}'$$

- If we reinject this into the first equation:

$$\begin{aligned} \begin{pmatrix} x' \\ y' \end{pmatrix} &= \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\ &\Leftrightarrow \begin{pmatrix} (\vec{e_x}')_x & (\vec{e_y}')_x \\ (\vec{e_x}')_y & (\vec{e_y}')_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \end{aligned}$$

Finding R comes back to finding $\vec{e_x}'$ and $\vec{e_y}'$

Similarities: how to ? – Central rotations (simple case) (3)



[CC BY L. Avanthey et al., 2018]

With **trigonometry** and the unit circle, it is **easy** to find the **coordinates** of **unit vectors** after **rotation**: $\vec{e_x}'$ and $\vec{e_y}'$

$$\begin{cases} \vec{e_x}' = \cos\Theta\vec{e_x} - \sin\Theta\vec{e_y} = \begin{pmatrix} \cos\Theta \\ -\sin\Theta \end{pmatrix} \\ \vec{e_y}' = \sin\Theta\vec{e_x} + \cos\Theta\vec{e_y} = \begin{pmatrix} \sin\Theta \\ \cos\Theta \end{pmatrix} \end{cases}$$

Similarities: how to ? – Central rotations (simple case) (4)

- Therefore:

$$R = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} = \begin{pmatrix} (\vec{e_x}')_x & (\vec{e_y}')_x \\ (\vec{e_x}')_y & (\vec{e_y}')_y \end{pmatrix} = \begin{pmatrix} \cos\Theta & \sin\Theta \\ -\sin\Theta & \cos\Theta \end{pmatrix}$$

Euclidian form :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\Theta & \sin\Theta \\ -\sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Similarities: Translation, Homothetie, Central Rotation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} T_x \\ T_y \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} T_x \\ T_y \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s \times x \\ s \times y \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s & 0 \\ 0 & s \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\Theta & \sin\Theta \\ -\sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\Theta & \sin\Theta \\ -\sin\Theta & \cos\Theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

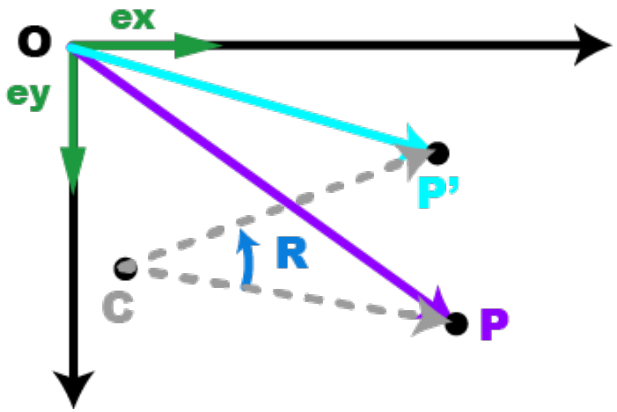
Towards a general Euclidean form of a similarity:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_x \\ b_y \end{pmatrix}$$

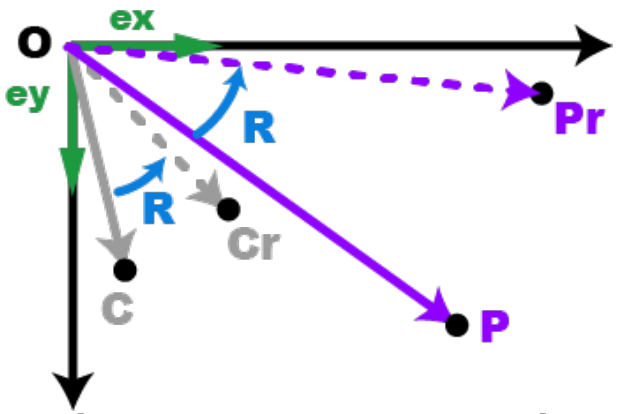
$$P' = AP + B$$

Non-central rotations (complexe case) (1)

In **general**, the **center** of **rotation** is **not located** on the **center** of the **reference**. But we **only know** how to do **central rotations**!



[CC BY NC SA L. Avanthey et al., 2018]



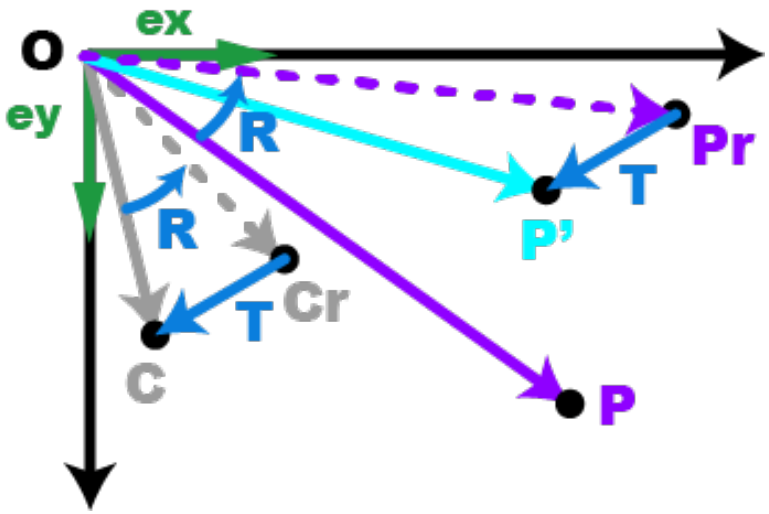
[CC BY NC SA L. Avanthey et al., 2018]

If we perform a **central rotation**, all points are moved, even our center of rotation while it is the only one to have to stay still!

How do we **correct** this?

Non-central rotations (complexe case) (2)

After performing a **central rotation**, we **notice** that we can **correct the offset** of the **real center** of rotation thanks to a **translation**!



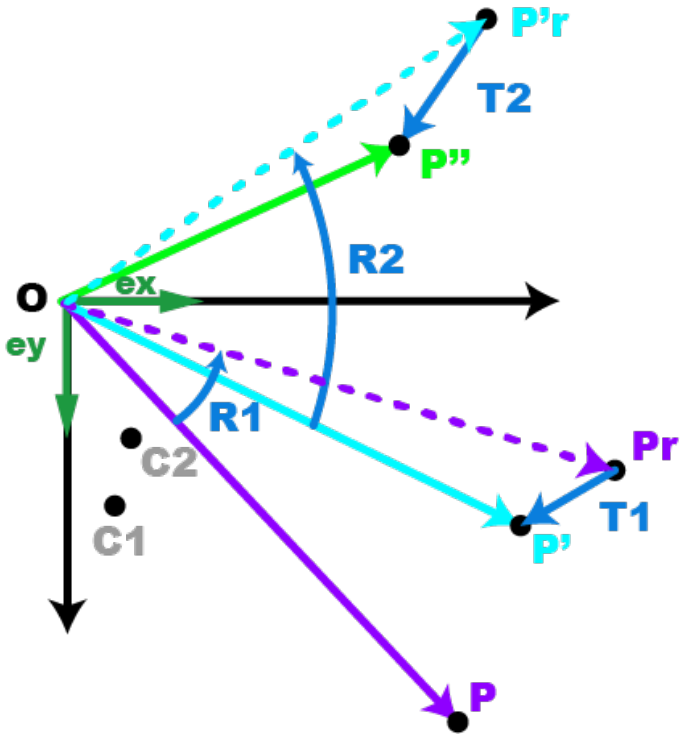
[L. Avanthey et al., 2018]

$\Rightarrow$ The **non-central rotation transformation** thus becomes:

$$P' = RP + T$$

Multiple non-central rotations

The **translations** are in **additions** and the **rotations** in **multiplications**: thus **chaining** non-central rotations quickly gives **complicated transformations**



[©①⊙ L. Avanthey et al., 2018]

- $P' = R_1 P + T_1$
- $P'' = R_2 P' + T_2$

$\Leftrightarrow \quad P'' = R_2 R_1 P + R_2 T_1 + T_2$

What can we do?

Because to **chain** hundreds of these transformations is an **every day task** in robotic vision (mosaics, 3D clouds, positioning, ...)

Homogeneous transformations (1)

A **similarity** $P' = AP + B$ can **also** be **written** in the form $P' = MP$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_x \\ b_y \end{pmatrix}$$

$$\Leftrightarrow$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & b_x \\ a_{10} & a_{11} & b_y \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

This is a **great step forward** to **simplify** the combination of transformations because we now have **only one matrix**: to **combine transformations** is thus **equivalent** to **multiply matrices!**

Homogeneous transformations (2)

The **previous expression** is **not entirely satisfactory** because of the **dysimetry** of $P' \begin{pmatrix} x' \\ y' \end{pmatrix}$ and $P \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

The **following expression**, called the **transform in homogenous coordinates**, is therefore **preferred**:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & b_x \\ a_{10} & a_{11} & b_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Bonus: the matrix is **square**, so it is potentially **invertible** (remember, we saw that it was wiser to use the inverse transformations, and we will see later that sometimes it is even necessary)!

Some homogeneous transformations

Homogeneous translation

$$T = \begin{pmatrix} 1 & 0 & Tx \\ 0 & 1 & Ty \\ 0 & 0 & 1 \end{pmatrix}$$

Homogeneous homothetie

$$S = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Homogeneous central rotation

$$R = \begin{pmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Homogeneous similarity

$$TSR = \begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Order is **important**:
$TSR \neq SRT$

$$\Leftrightarrow TSR = \begin{pmatrix} s\cos\Theta & s\sin\Theta & T_x \\ -s\sin\Theta & s\cos\Theta & T_y \\ 0 & 0 & 1 \end{pmatrix}$$

The same... but in inverse transformations

$$T^{-1} = \begin{pmatrix} 1 & 0 & -Tx \\ 0 & 1 & -Ty \\ 0 & 0 & 1 \end{pmatrix}$$

$$S^{-1} = \begin{pmatrix} \frac{1}{s} & 0 & 0 \\ 0 & \frac{1}{s} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

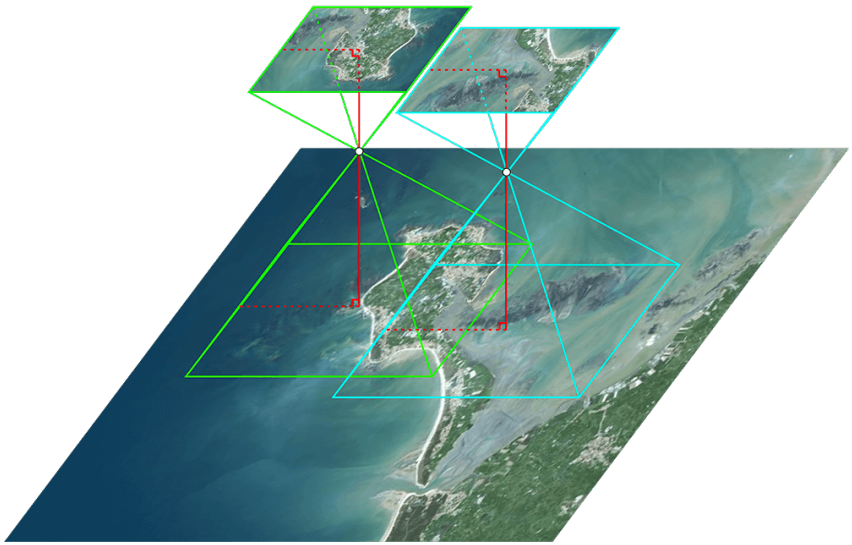
$$R^{-1} = \begin{pmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$(TSR)^{-1} = R^{-1}S^{-1}T^{-1} = \begin{pmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{s} & 0 & 0 \\ 0 & \frac{1}{s} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -T_x \\ 0 & 1 & -T_y \\ 0 & 0 & 1 \end{pmatrix}$$

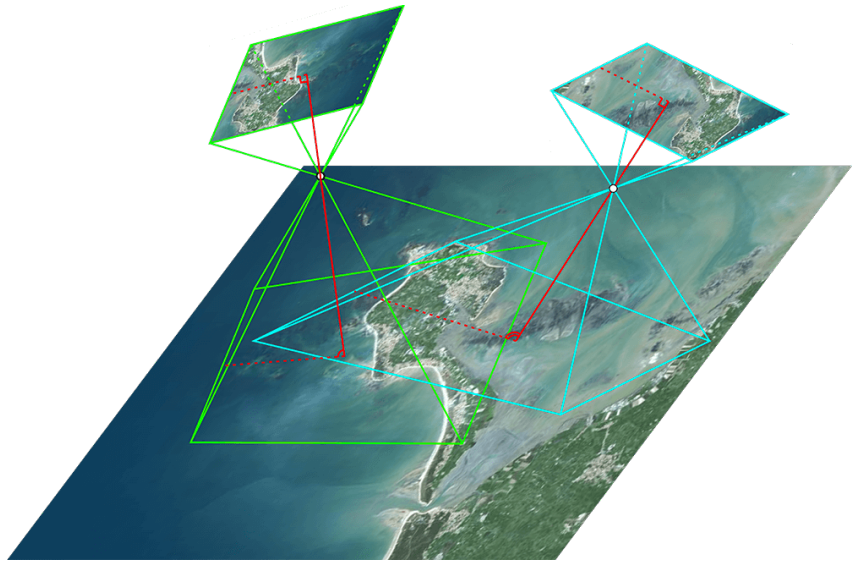
$$\Leftrightarrow (TSR)^{-1} = \begin{pmatrix} \frac{\cos(\Theta)}{s} & -\frac{\sin(\Theta)}{s} & \frac{-T_x\cos(\Theta)+T_y\sin(\Theta)}{s} \\ \frac{\sin(\Theta)}{s} & \frac{\cos(\Theta)}{s} & \frac{-T_x\sin(\Theta)-T_y\cos(\Theta)}{s} \\ 0 & 0 & 1 \end{pmatrix}$$

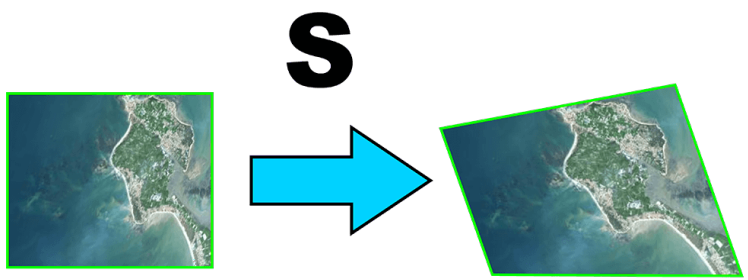
Similarities(?): registrations (general case)

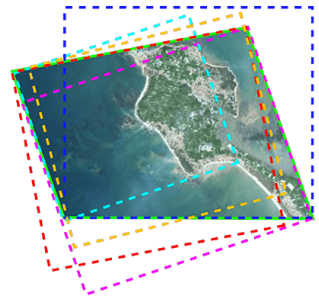
Simple case
(fronto-parallel shots)



General case
(multi-orientation shots)







[CC BY L. Avanthey et al., 2020]

Affinities(?): registrations (general case) (1)

We need more **degrees of freedom** (DoF):
there is 4 DoF only for a similarity (T_x, T_y, θ, s), let's try with the **affinities** which add **2 more** DoF

The **Euclidean form** of an **affinity** is the **same** as the **general form** of a **similarity**

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & b_x \\ a_{10} & a_{11} & b_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Affinities: how to? – Aspect ratio

The first degree of freedom concerns the **aspect ratio**

Unlike with similarity, the **scale** is no longer **homogeneous**

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

[CC BY NC SA L. Avanthey et al., 2020]

Affinities: how to ? – Shear

The second degrees of freedom concerns the **shear** (or skew)

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

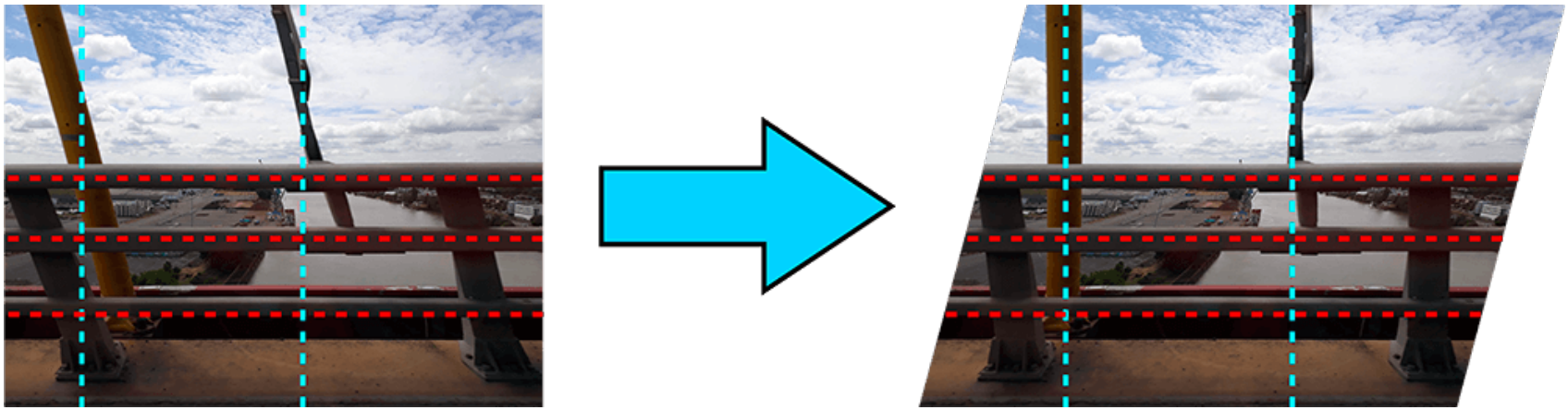
[CC BY NC SA L. Avanthey et al., 2020]

In our applications,

when do you think we end up with this type of transformation other than in relation to the position of the camera?

Shear & rolling shutter (1)

The **rolling shutter**, in the presence of a **constant speed** of displacement, causes a deformation which can be **roughly corrected** by a **shear** transformation.

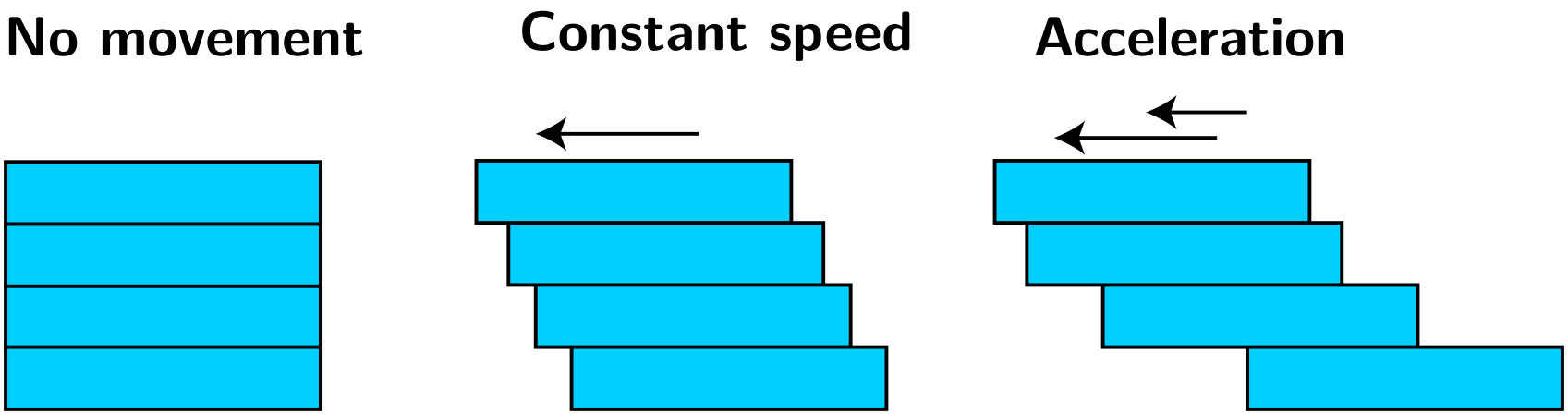


[L. Avanthey et al., 2020]

The **posts** of the railing, the lamppost and the sign are **not vertical** on the shoot because of the **rolling shutter** and the **displacement**. Take the **test** with your **cellphone** device on the **highway** and compare with what your **eyes** see!

Shear & rolling shutter (2)

On the other hand, if the **speed** is **not constant** (or if the relative speed is not constant, which is the case when there are close and distant objects), one **cannot correct** the image with **shear** which is a **global geometric transformation**.



[L. Avanthey et al., 2020]

Shear can also be useful for registering the bands of an image acquired by a **pushbroom satellite**.

Affinities(?): registrations (general case) (2)

6 DoF is better but still **insufficient** to register our images in the general case.

[L. Avanthey et al., 2020]

The **affinities** preserve **parallelism**, however, the **perspective transformation** (to move from the 3D world to the 2D plane of the image) has undermined this property.

We need **more degrees of freedom**.

Homographies(?): registrations (general case)

The **homographic transformation** allows us to **add two degrees of freedom** (for a total of 8)

$\Rightarrow$ It is done by playing with the **two zero coefficients** of the matrix of the **affine transformation**.

Warning, if these coefficients are **no longer zero**, the **homogeneous term** of our point coordinates will be **modified**

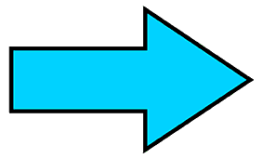
$\Rightarrow$ We must therefore return to the **homogeneous coordinates** after applying the transformation.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{u}{w} \\ \frac{v}{w} \\ \frac{w}{w} \end{pmatrix} \text{ such that: } \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{11} \\ a_{20} & a_{21} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Homographies: how to ? Hinge

These two additional degrees of freedom are expressed by the **horizontal** and **vertical** hinge coefficients.

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ h_x & h_y & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$



[L. Avanthey et al., 2020]

Combined with the shear coefficient, it allows in particular to correct the **keystone** effect of video projectors

Homographies (?): registrations (general case)

Will these **8 DoF** be **enough** to register our images in the **general case**?

⇒ By **combining shear** and **hinge** in addition to **rotations**, **translations** and **scale**: we get there!

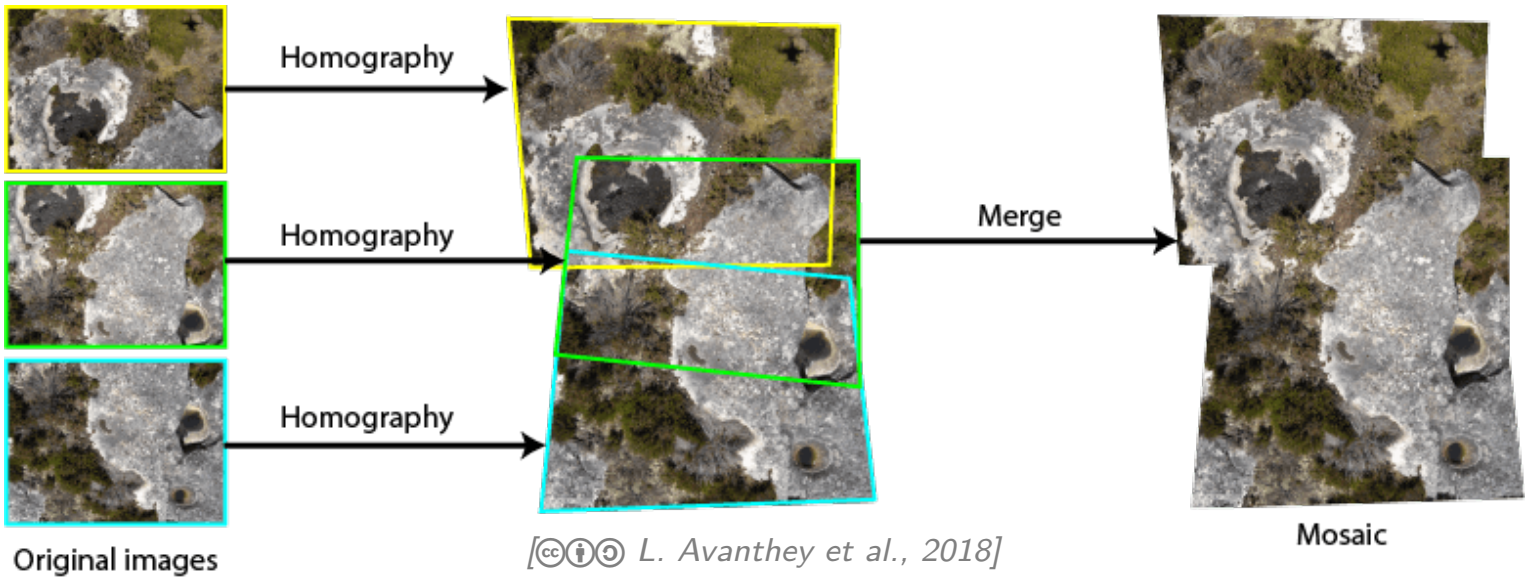
[L. Avanthey et al., 2020]

Homographies seem to be our **solution**!

Homographies (!): registrations (general case)

In the **general case**, the perspective transformation has broken the **parallelism** of the scene: a **homographic transformation**, with its **8 DoF**, can model these deformations without going back to 3D!

If we know the **coordinates** of **4 corresponding points** between the two images, we obtain the 8 equations necessary to solve the system and find the **homography** which connects them.



[L. Avanthey et al., 2018]

Homographies: all registration cases?

In your opinion, can **homographies** describe **all the general cases** of registration?

Homographies: not all registration cases!

The problem comes from the scenes which present **relief**: as for the rolling shutter with different relative speeds, the **geometric deformations** will no longer be **global** within the same image!

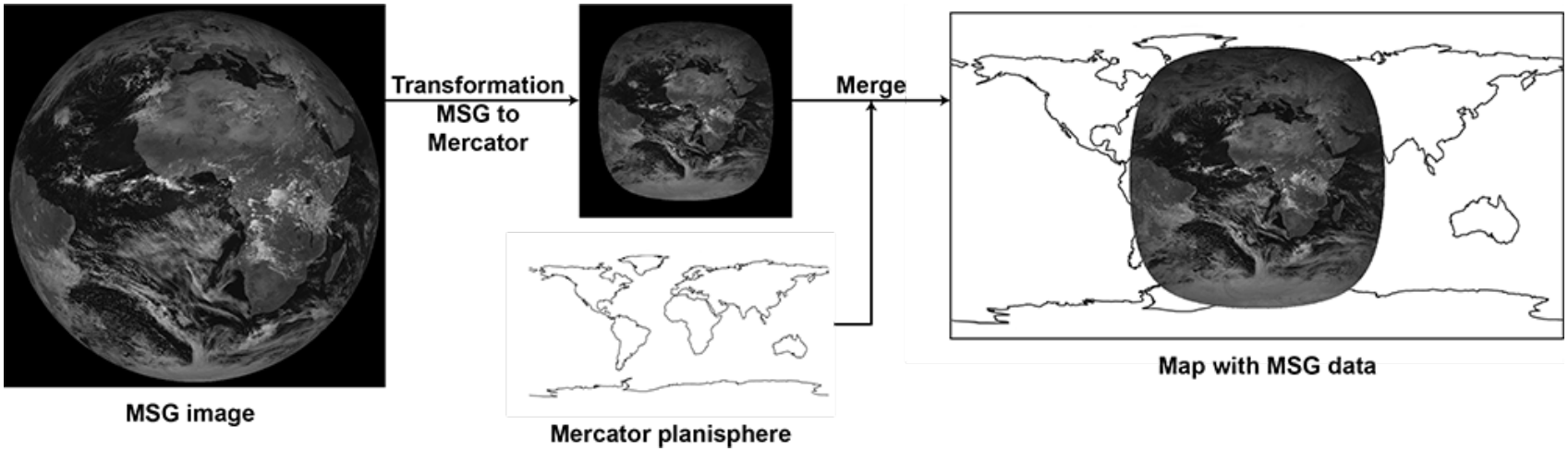
So we manage to have **almost plannar scenes** when we want to make **mosaics**.

⇒ This is often the case in **aerial applications** because you can afford to **fly quite high**, unless you need **extreme precision** on a **very rough scene**.

⇒ On the other hand in **underwater applications**, one must be **close** to the scene because of the **absorption**, and the **relief** is then generally **too strong**: one must in these cases go back through the **3D geometry** to be able to build a **mosaic**.

Exotic transformations: Mercator projection

To **represent** the image data on a **map background** (planisphere) we use a **complex transformation**. In the **Mercator model**, it is a **cylindrical projection** tangent to the **equator** of the globe on a **plane**



[L. Avanthey et al., 2018]

Mercator projection **preserves angles** but **distorts distances** and **areas**