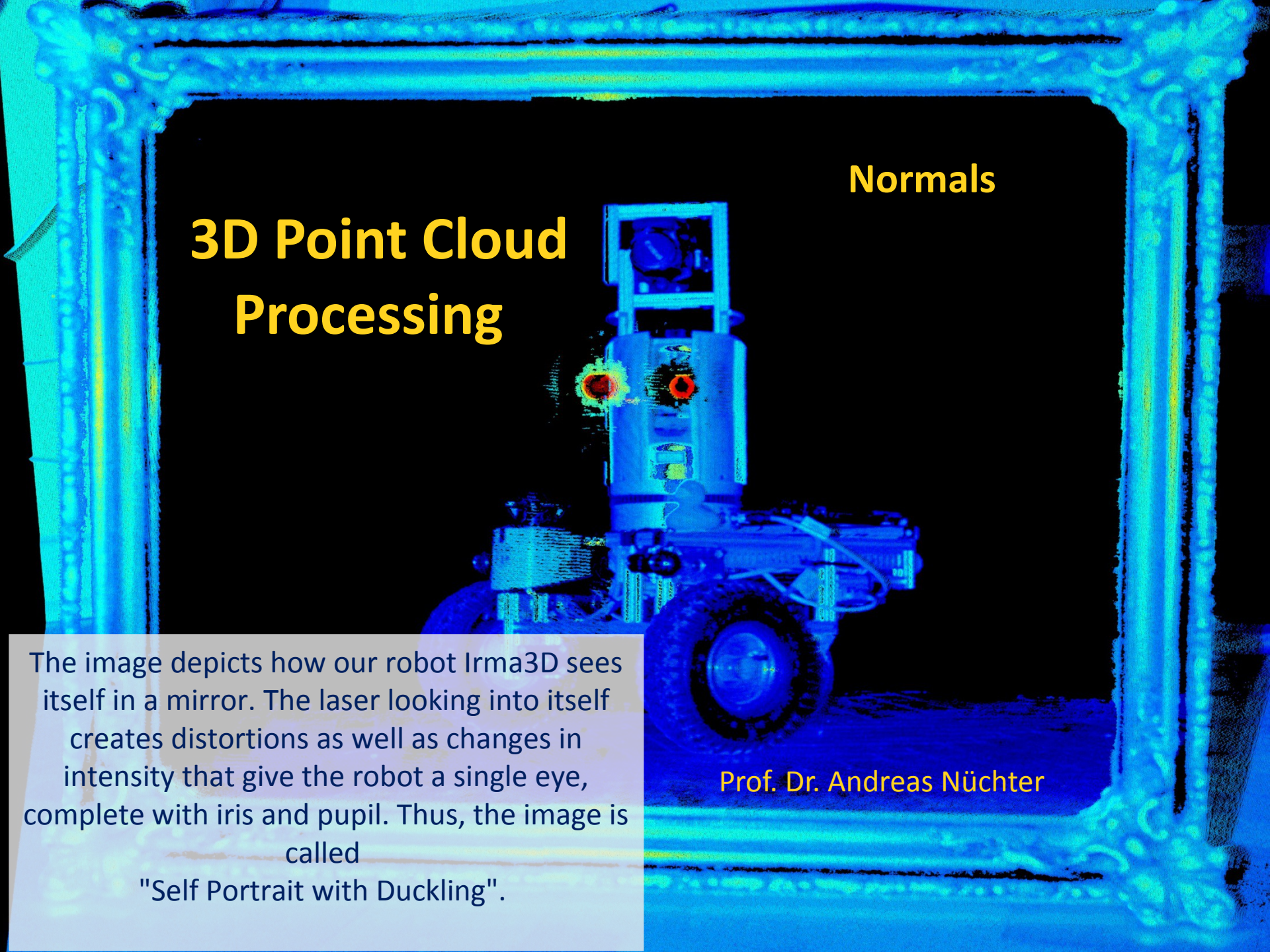
# 3D Point Cloud Processing

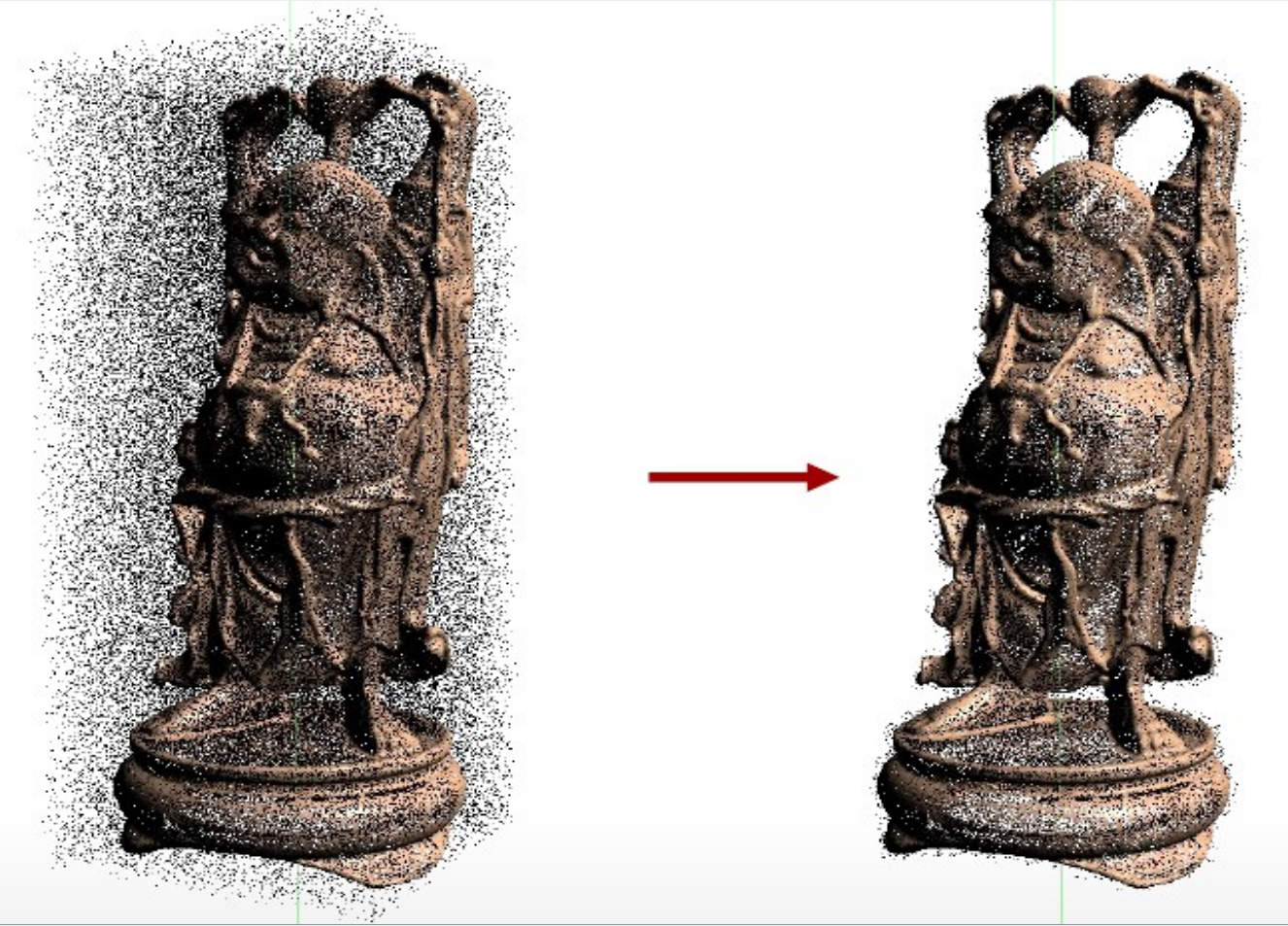


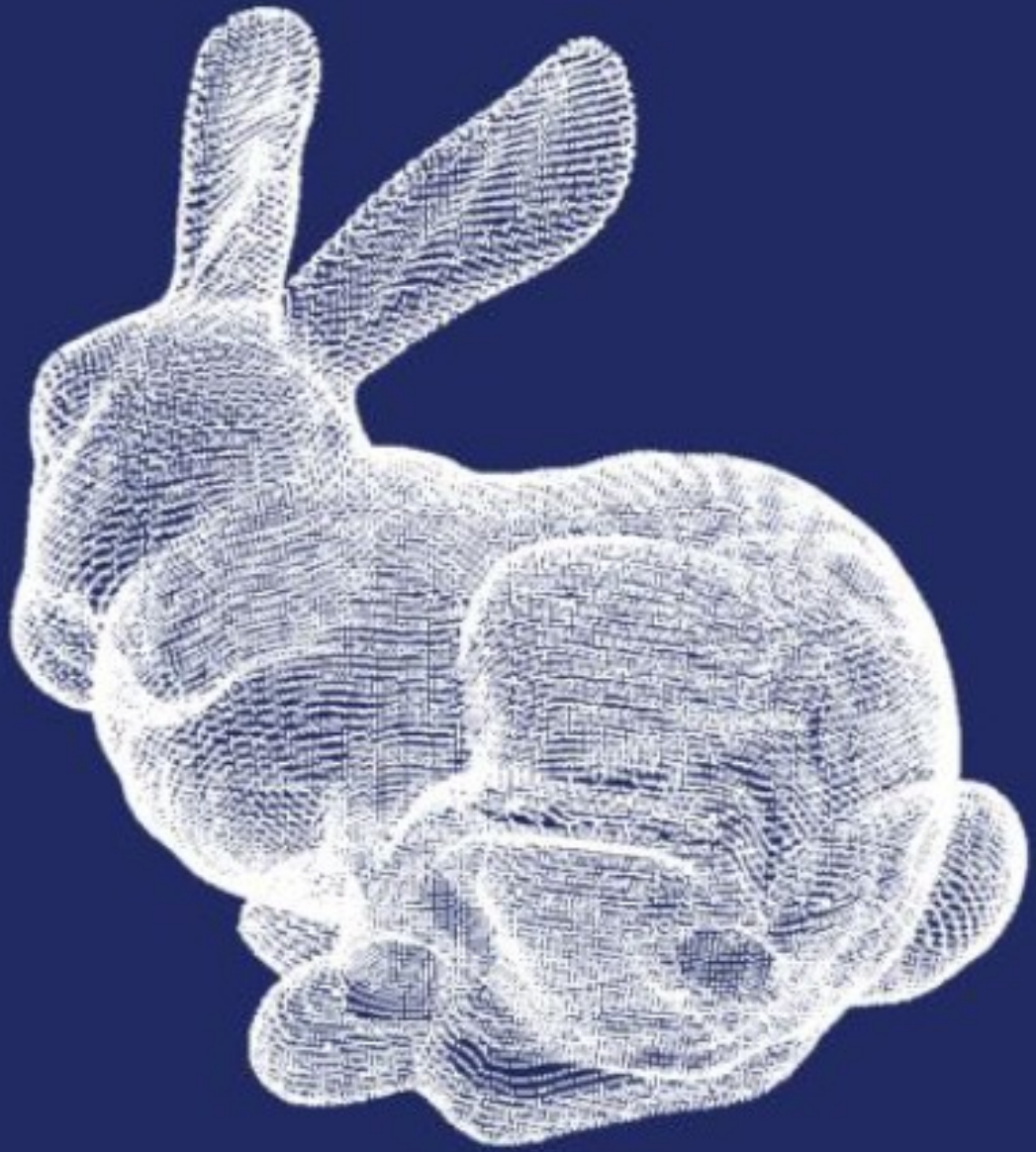

The image depicts how our robot Irma3D sees itself in a mirror. The laser looking into itself creates distortions as well as changes in intensity that give the robot a single eye, complete with iris and pupil. Thus, the image is called
"Self Portrait with Duckling".

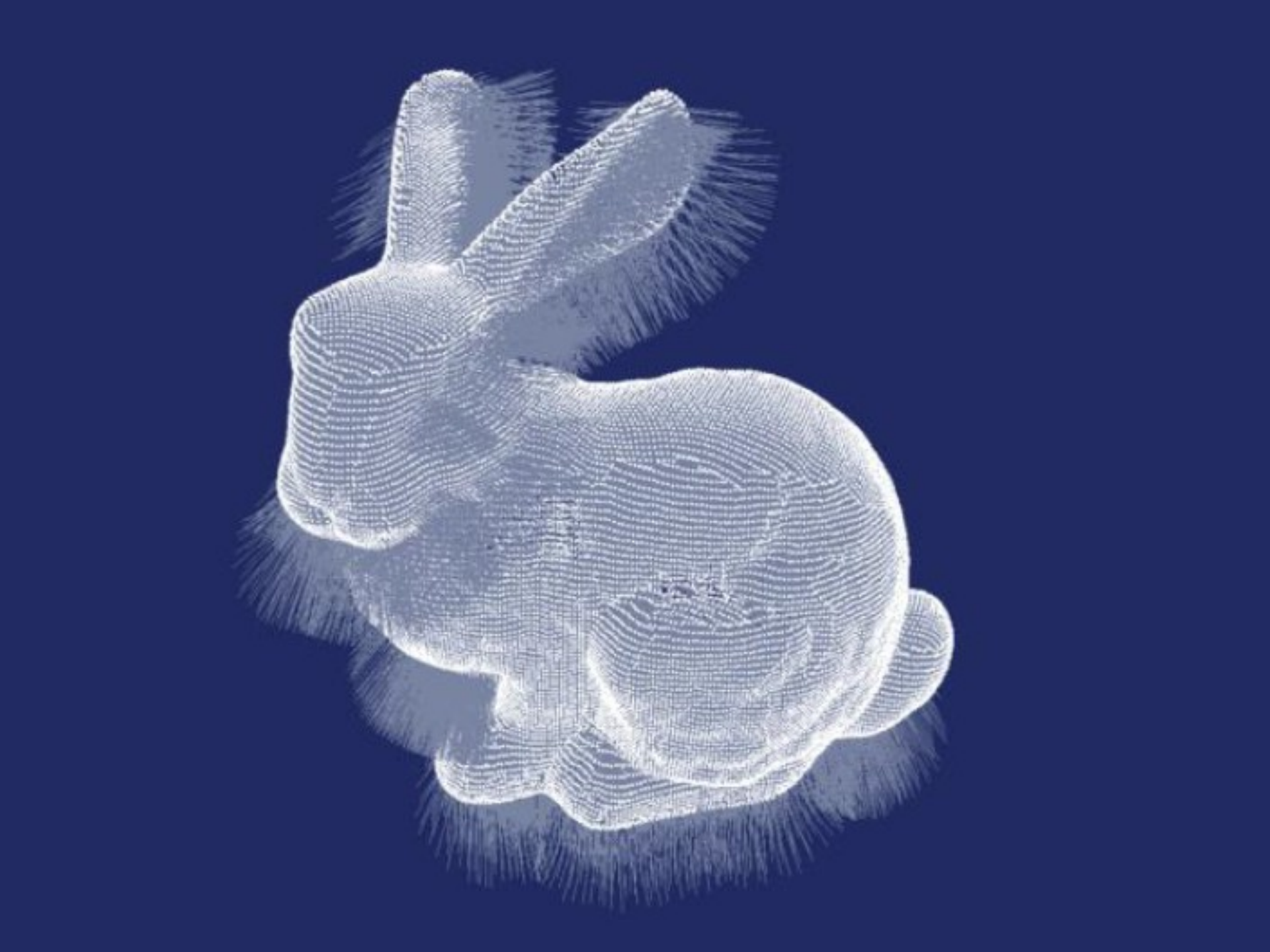
Prof. Dr. Andreas Nüchter

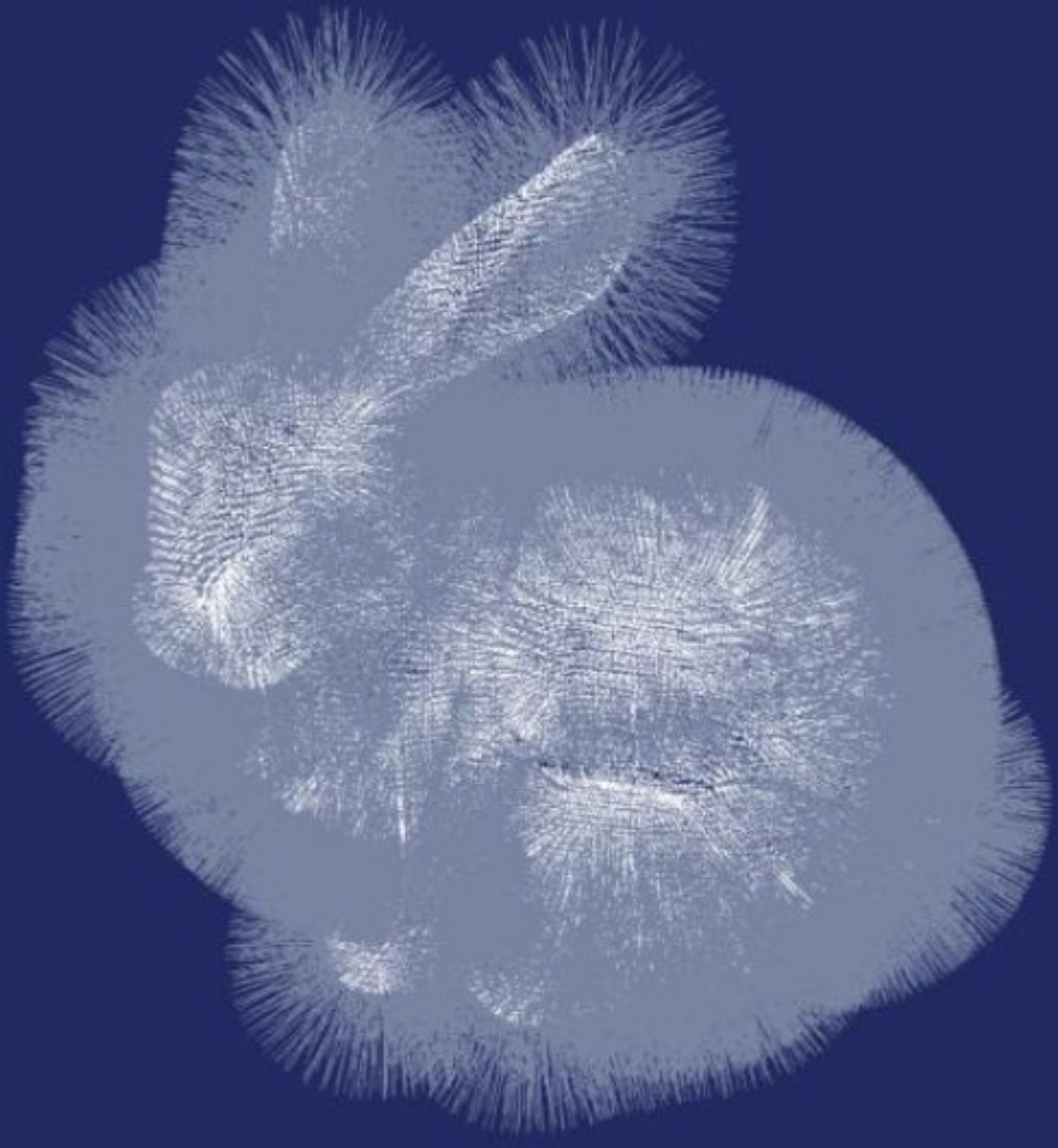
# Outlier Removal

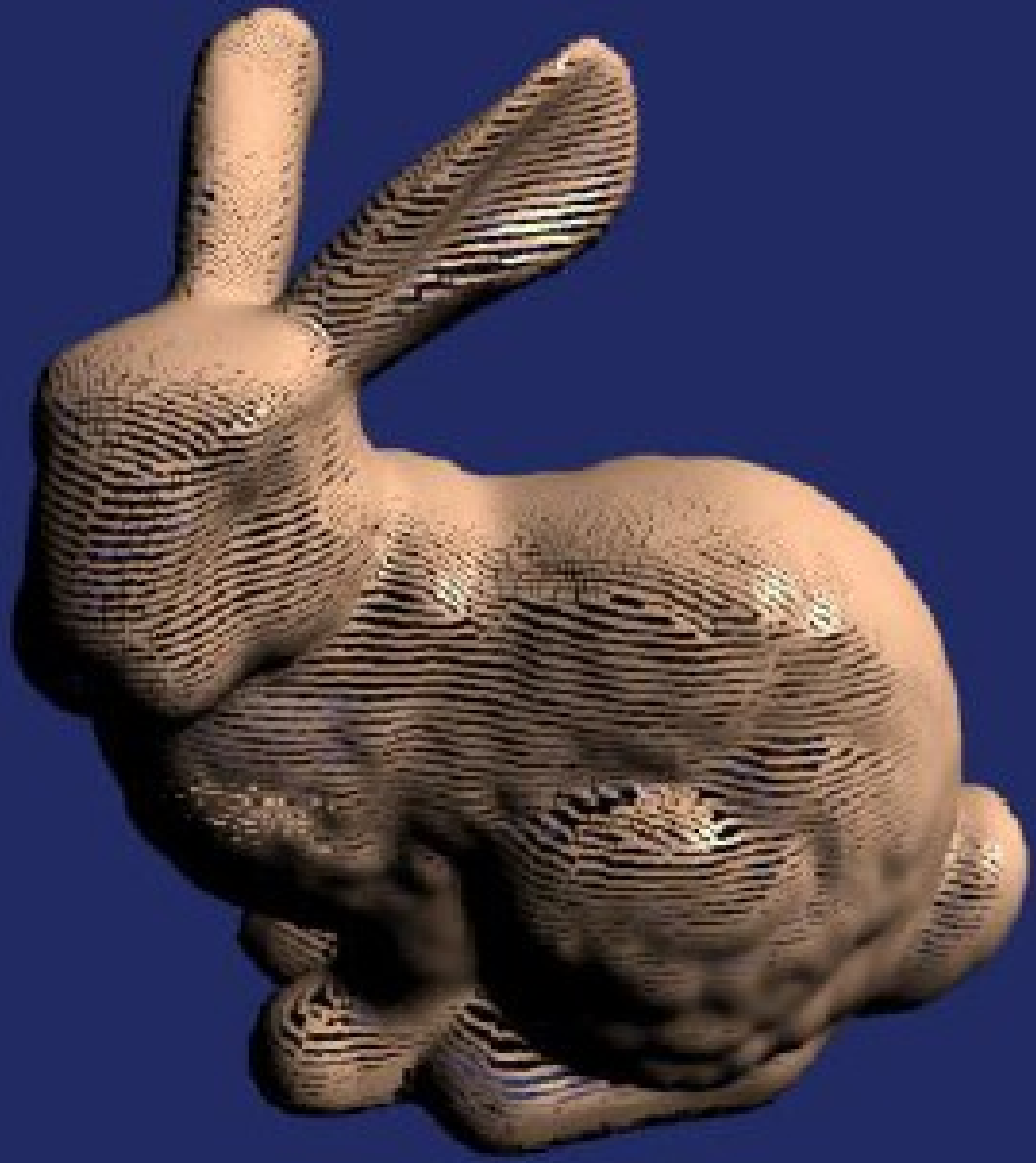
- 3D laser scans contain noise
  - Especially at so-called jump edges



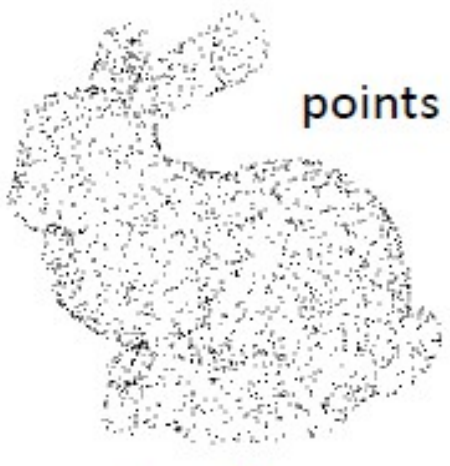








# Application in CG: Direct Point Splatting



# Principal Component Analysis

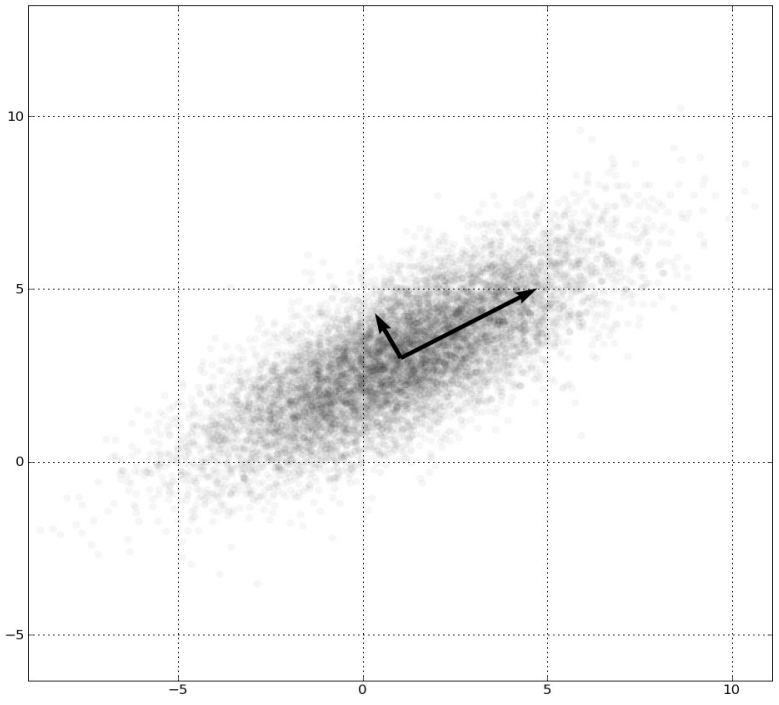
- PCA was invented in 1901 by Karl Pearson.
- PCA can be done by eigenvalue decomposition of a data covariance (or correlation) matrix

  or

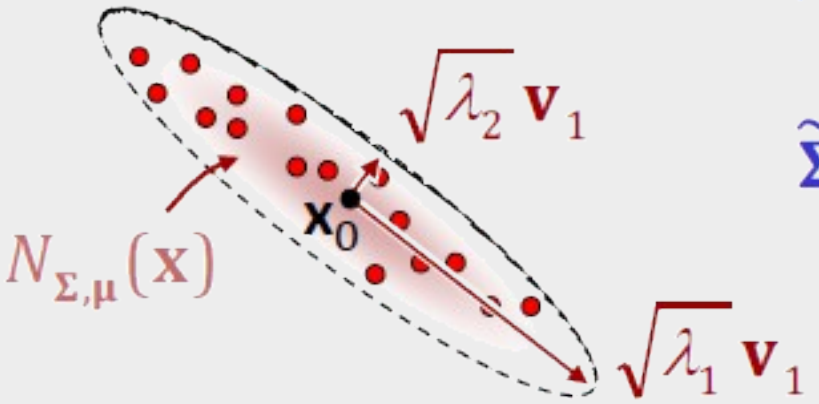
  singular value decomposition of a data matrix, usually after mean centering (and normalizing) the data matrix for each attribute.
- “PCA is the simplest of the true eigenvector-based multivariate analysis. Often, its operation can be thought of as revealing the internal structure of the data in a way that best explains the variance in the data. If a multi-variate data set is visualized as a set of coordinates in a high-dimensional data space (1 axis per variable), PCA can supply the user with a lower-dimensional picture, a "shadow" of this object when viewed from its (in some sense) most informative viewpoint. This is done by using only the first few principal components so that the dimensionality of the transformed data is reduced.” [wikipedia]


# Principal Component Analysis

- “PCA of a multivariate Gaussian distribution centered at (1,3) with a standard deviation of 3 in roughly the (0.878, 0.478) direction and of 1 in the orthogonal direction. The vectors shown are the eigenvectors of the covariance matrix scaled by the square root of the corresponding eigen-value, and shifted so their tails are at the mean.” [wikipedia]



# PCA Plane Fitting




$$\boldsymbol{\mu} = \mathbf{x}_0 = \frac{1}{n}\sum_{i=1}^{n} d_i$$

$$\tilde{\boldsymbol{\Sigma}} = \frac{1}{n-1}\mathbf{S} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{d}_i - \mathbf{x}_0)(\mathbf{d}_i - \mathbf{x}_0)^{\mathrm{T}}$$

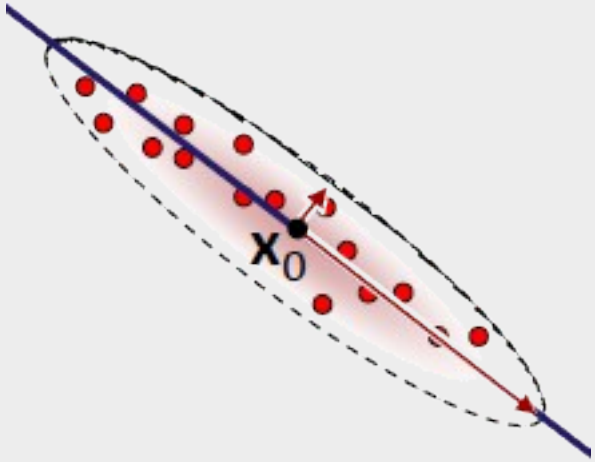
$$N_{\Sigma,\mu}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\det(\boldsymbol{\Sigma})^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

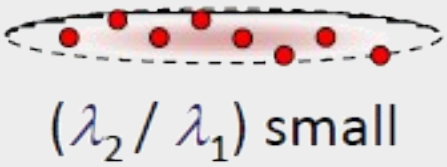
- PCA can be interpreted as fitting a Gaussian distribution and computing the main axes

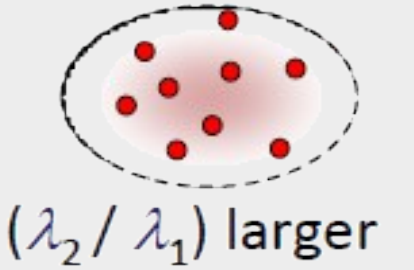
# PCA Plane Fitting

**Plane Fitting in 3D:**

- Sample mean and the two directions of maximum eigenvalues
- Smallest eigenvalue
  - Eigenvector points in normal direction
  - Aspect ratio ($l_3 / l_2$) is a measure of “flatness” (quality of fit)
- Total least squares optimal *normal direction* (up to sign) given by eigenvector with smallest eigenvalue



# Algorithm Outliers Removal

## Very simple outliers removal algorithm:

- For each point compute its k nearest neighbors, e.g., k=20
- Compute the principal component analysis (plane fit with total least squares)
- If the *third* eigenvalue (normal direction) is larger than 1 / ( 1+epsilon ) times the *second* eigenvalue, delete the point as an outlier

# Algorithm Normal Computation (1)

## Very simple normal computation algorithm:

- For each point compute its k nearest neighbors, e.g., k=20
- Compute the principal component analysis (plane fit with total least squares)
- The smallest eigenvector is your normal vector. Scale it to length 1. Orient it towards the scan pose.


# Algorithm Normal Computation (2)

- The choice of *k* is essential for the quality of the calculated normal.
- The smaller the value of *k*, the lower the needed processing time, since fever tree traversals are necessary. However, low *k* values are sensitive to noise.
- Higher values may compensate sensor noise in the approximation process, but they increase the processing time and might lead to wrong results, because sharp features will be “smoothed” out.

- Handling real 3D laser scanner data as input shows another difficulty: With increasing object distance the point density decreases.

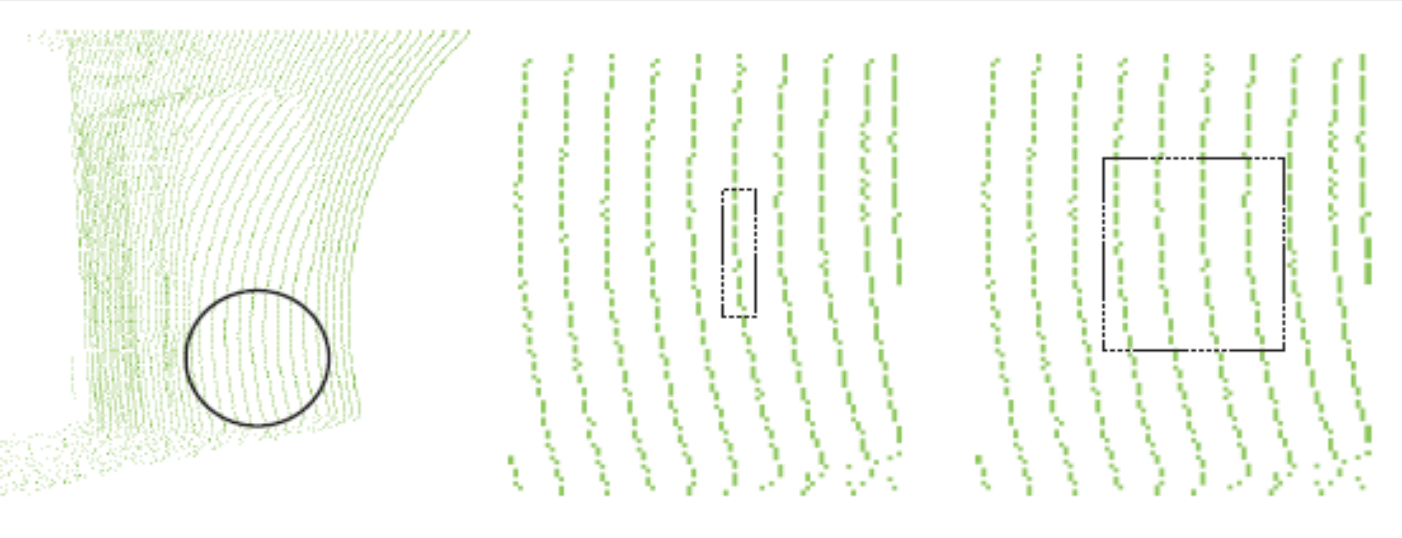
# Algorithm Normal Computation (2)

- The choice of *k* is essential for the quality of the calculated normal.
- The smaller the value of *k*, the lower the needed processing time, since fever tree traversals are necessary. However, low *k* values are sensitive to noise.
- Higher values may compensate sensor noise in the approximation process, but they increase the processing time and might lead to wrong results, because sharp features will be “smoothed” out.

- Handling real 3D laser scanner data as input shows another difficulty: With increasing object distance the point density decreases.

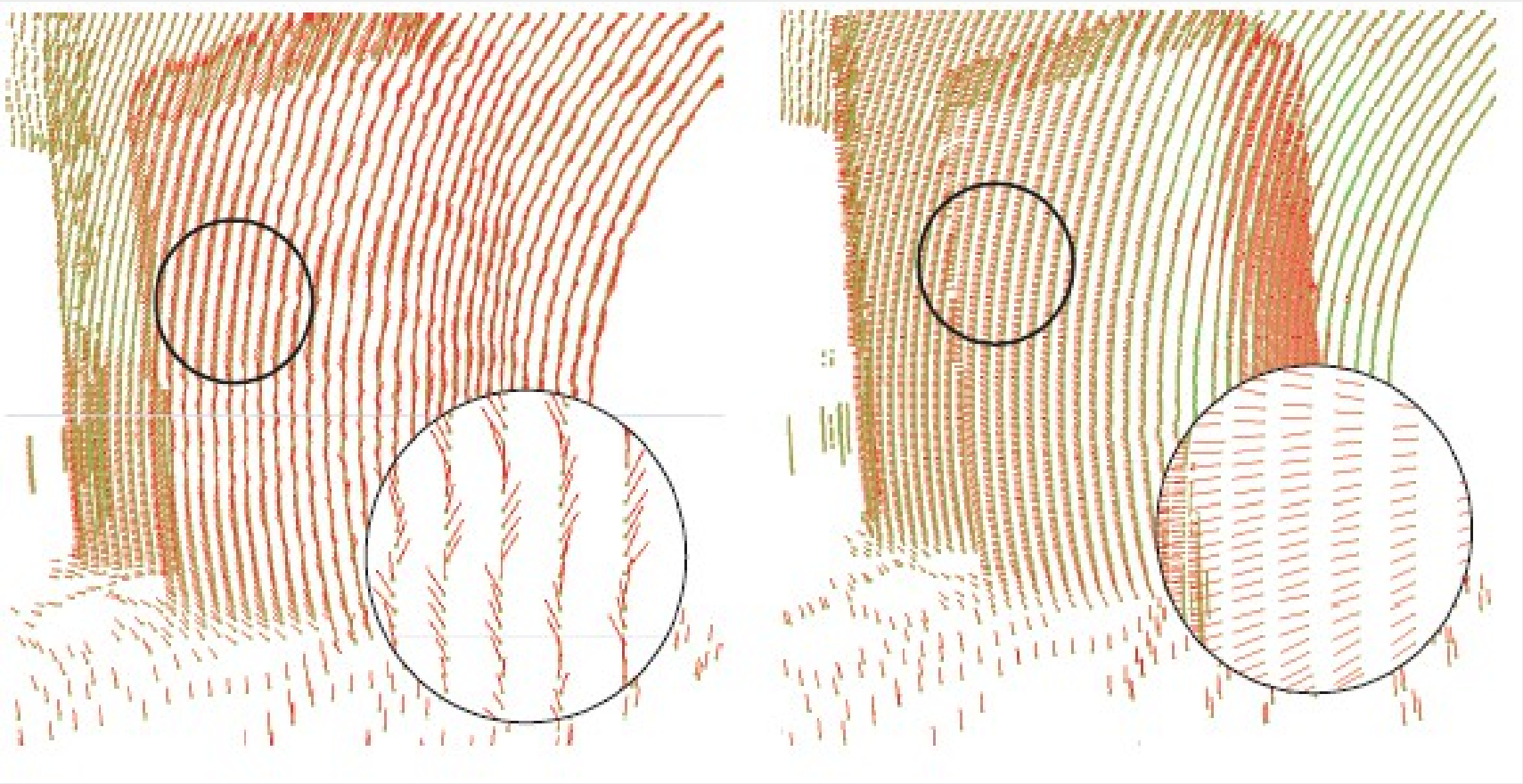
# Algorithm Normal Computation (3)

- Solution: Adapt *k* dynamically to the data density.
  - To detect ill formed *k*-neighborhoods, analyze the shape of their bounding boxes.
  - Critical configurations will result in elongate bounding boxes.
  - If such a configuration is detected, *k* is increased until this shape criterion is fulfilled.



# Algorithm Normal Computation (4)

- Results *k*-adaptation



# Total Least Square Method (1)

- Given *k* points in the neighborhood
- Find $\mathbf{n} = (n_x, n_y, n_z)^T$

such that

$$e = \sum_{i=1}^{k} (\mathbf{p}_i^T \mathbf{n} - d)^2 \text{ is minimum subject to } |\mathbf{n}| = 1.$$

Please note the plane equation

$$n_x x + n_y y + n_z z - d = 0$$

The minimum is given by

$$M = \frac{1}{k} \sum_{i=1}^{k} (\mathbf{p}_i - \overline{\mathbf{p}})(\mathbf{p}_i - \overline{\mathbf{p}})^T \text{ where } \overline{\mathbf{p}} = \frac{1}{k} \sum_{i=1}^{k} \mathbf{p}_i$$



# Total Least Square Method (2)

- Runtime

  N times *k*-NN time
  - O (N^2) if we use linear search for closest poitns
  - O (N log N) if we use something like k-NN using the library we used in assignment #9
  - O(N) if we use something clever

- Use panorama images!
- Approximate the closest points by looking into the neighboring pixels (extended Map)

- Further drawbacks
  - For every point, we have to calculate the eigenvalues of a 3x3 matrix.
  - Solving the characteristic polynomial of the degree 3


# Once again: Total Least Square Method

- Given *k* points in the neighborhood
- Find $\mathbf{n} = (n_x, n_y, n_z)^T$

such that

$$e = \sum_{i=1}^{k} (\mathbf{p}_i^T \mathbf{n} - d)^2 \text{ is minimum subject to } |\mathbf{n}| = 1.$$

Please note the plane equation

$$n_x x + n_y y + n_z z - d = 0$$

The minimum is given by

$$M = \frac{1}{k} \sum_{i=1}^{k} (\mathbf{p}_i - \overline{\mathbf{p}})(\mathbf{p}_i - \overline{\mathbf{p}})^T \text{ where } \overline{\mathbf{p}} = \frac{1}{k} \sum_{i=1}^{k} \mathbf{p}_i$$


# Panorama Images for Computing Normals (1)

$$e = \sum_{i=1}^{k} (\mathbf{p}_i^T \mathbf{n} - d)^2 \text{ is minimum subject to } |\mathbf{n}| = 1.$$

- Differentiate by $d^2$ and and further division by the squared range $\rho_i^2$ (considering that the neighborhood is small and therefore the ranges are similar and can be assumed equal - this is the key approximation made) yields:

$$e = \sum_{i=1}^{k} ((\rho^{-1}\mathbf{v}_i)^T \widehat{\mathbf{n}} - \rho_i^{-1})^2$$

$$\mathbf{v}_i = \begin{bmatrix} \cos\theta_i \sin\phi_i \\ \sin\theta_i \sin\phi_i \\ \cos\phi_i \end{bmatrix}$$



# Panorama Images for Computing Normals (2)

- This suggests a solution for $\hat{\mathbf{n}}$ as:

$$\hat{\mathbf{n}} = \widehat{M}^{-1} b$$

where

$$\widehat{M} = \sum_{i=1}^{k} \mathbf{v}_i \mathbf{v}_i^T, \; b = \sum_{i=1}^{k} \frac{\mathbf{v}_i}{\rho_i}$$

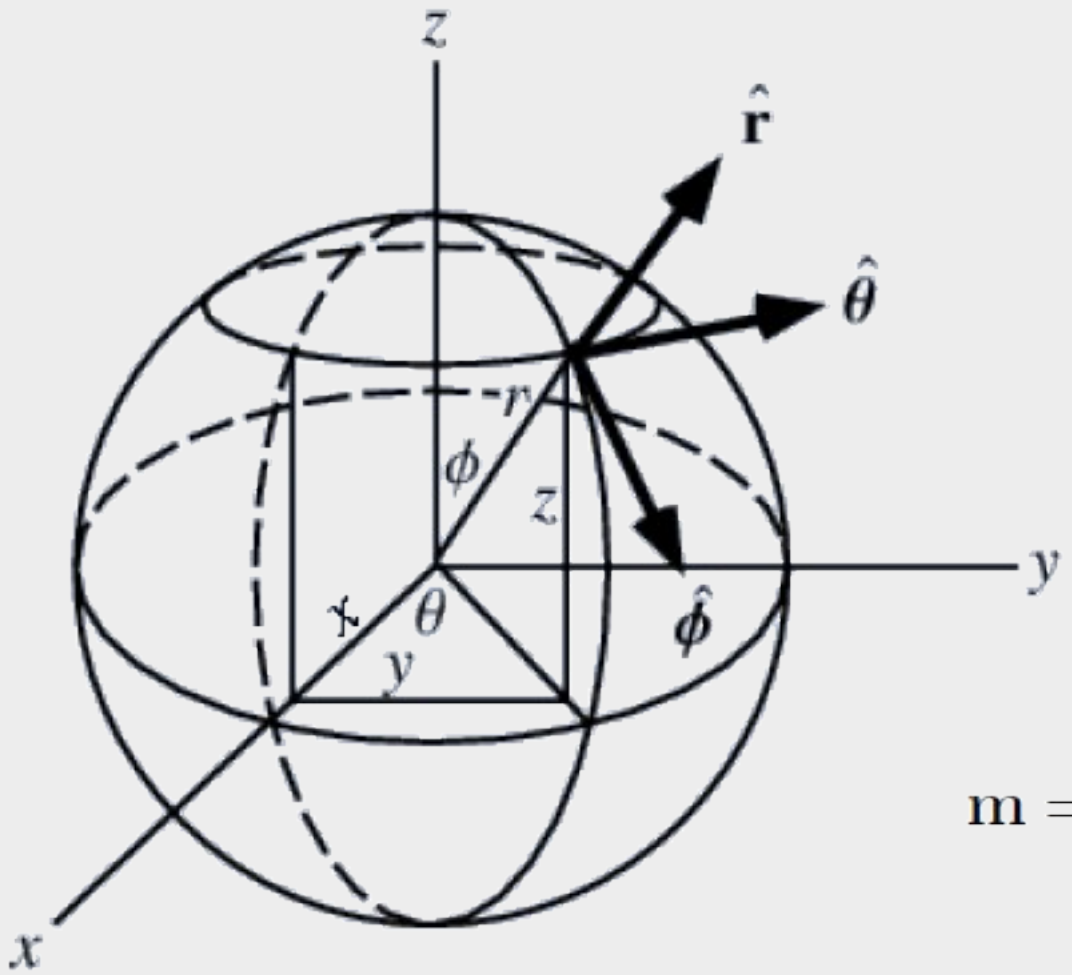
- This way the computation of eigenvalues is avoided and the matrix $\widehat{M}$ can be precomputed for the desired image coordinates as it does not depend on the range. Also, this approximation works directly on spherical coordinates and no overhead from coordinate system conversion is present.

H. Badino, D. Huber, Y. Park, and T. Kanade. Fast and Accurate Computation of Surface Normals from Range Images. IEEE International Conference on Robotics and Automation, 11:3084 { 3091, May 2011.



# Spherical Coordinates (1)

- Spherical coordinates




$$\mathbf{v} = \rho \begin{bmatrix} \cos\theta \sin\phi \\ \sin\theta \sin\phi \\ \cos\phi \end{bmatrix}$$

$$\mathbf{m} = \begin{bmatrix} \rho \\ \theta \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2 + z^2} \\ \arctan(y/x) \\ \arcsin(z/\rho) \end{bmatrix}$$

# Spherical Coordinates (2)

- Partial derivatives

$$\nabla \equiv \vec{x}\frac{\partial}{\partial x} + \vec{y}\frac{\partial}{\partial y} + \vec{z}\frac{\partial}{\partial z}$$

using the unit vectors along the x/y/z-axis

- Differentiating

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \rho\cos\theta\sin\phi \\ \rho\sin\theta\sin\phi \\ \rho\cos\phi \end{bmatrix}$$

yields

# Spherical Coordinates (3)

$$\begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} = \begin{bmatrix} \cos\theta\sin\phi d\rho - \rho\sin\theta\sin\phi d\theta + \rho\cos\theta\cos\phi d\phi \\ \sin\theta\sin\phi d\rho + \rho\cos\theta\sin\phi d\theta + \rho\sin\theta\cos\phi d\phi \\ \cos\phi d\rho - \rho\sin\phi d\phi \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta\sin\phi & -\rho\sin\theta\sin\phi & \rho\cos\theta\cos\phi \\ \sin\theta\sin\phi & \rho\cos\theta\sin\phi & \rho\sin\theta\cos\phi \\ \cos\phi & 0 & -\rho\sin\phi \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix}$$

- or

$$\begin{bmatrix} d\rho \\ d\theta \\ d\phi \end{bmatrix} = \begin{bmatrix} \cos\theta\sin\phi & \sin\theta\sin\phi & \cos\phi \\ -\dfrac{\sin\theta}{\rho\sin\phi} & \dfrac{\cos\theta}{\rho\sin\phi} & 0 \\ \dfrac{\cos\theta\cos\phi}{\rho} & \dfrac{\sin\theta\cos\phi}{\rho} & -\dfrac{\sin\phi}{\rho} \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix}$$

# Spherical Coordinates (4)

- And finally

$$\nabla \equiv \vec{x}\left(\cos\theta\cos\phi\frac{\partial}{\partial\rho} - \frac{\sin\theta}{\rho\sin\phi}\frac{\partial}{\partial\theta} + \frac{\cos\theta\cos\phi}{\rho}\frac{\partial}{\partial\phi}\right) +$$

$$\vec{y}\left(\sin\theta\cos\phi\frac{\partial}{\partial\rho} + \frac{\cos\theta}{\rho\sin\phi}\frac{\partial}{\partial\theta} + \frac{\sin\theta\cos\phi}{\rho}\frac{\partial}{\partial\phi}\right) +$$

$$\vec{z}\left(\cos\phi\frac{\partial}{\partial\rho} - \frac{\sin\phi}{\rho}\frac{\partial}{\partial\phi}\right)$$

# Panorama Images for Computing Normals (3)

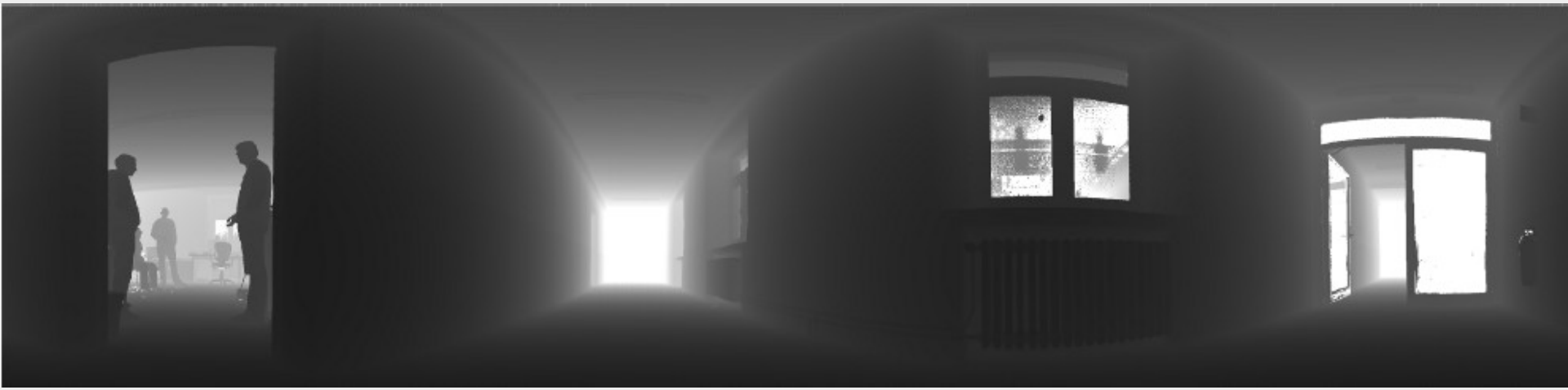
- We want to compute

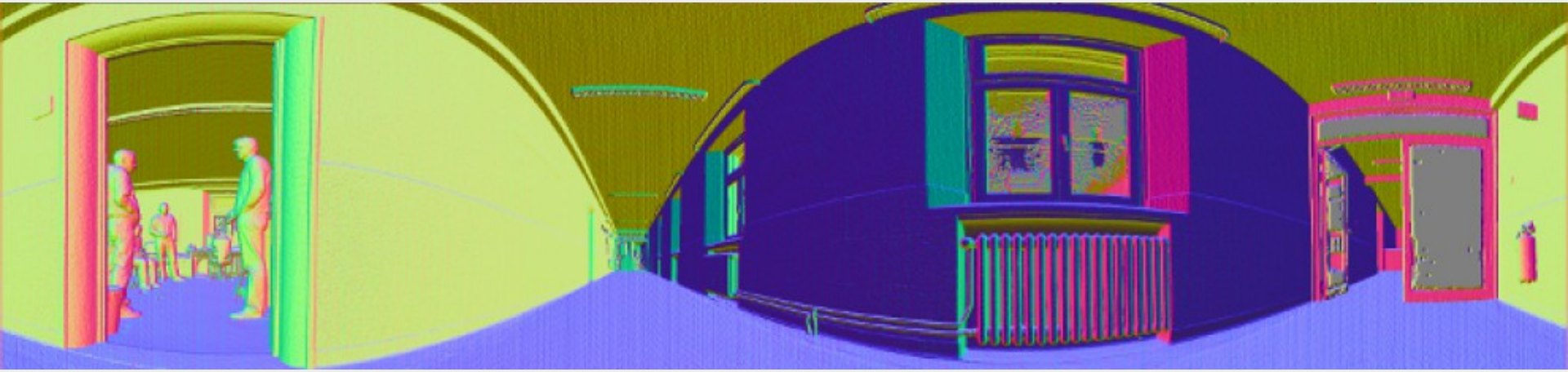
$$\mathbf{n} = \nabla\rho = \nabla s(\theta, \phi)$$

- And therefore

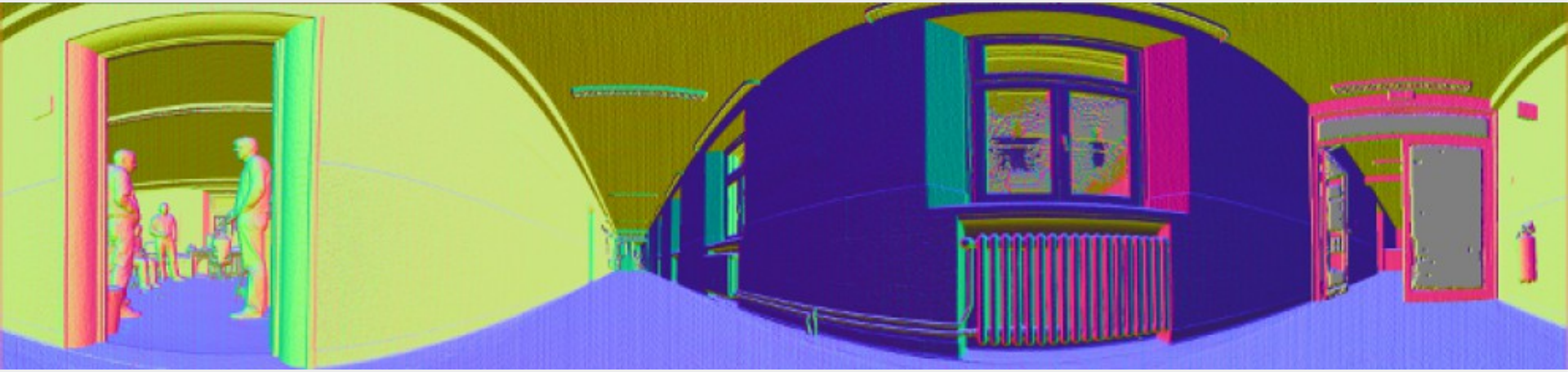
$$\mathbf{n} = \begin{bmatrix} \cos\theta\sin\phi - \dfrac{\sin\theta}{\rho\sin\phi}\dfrac{\partial\rho}{\partial\theta} + \dfrac{\cos\theta\cos\phi}{\rho}\dfrac{\partial\rho}{\partial\phi} \\ \sin\theta\sin\phi - \dfrac{\cos\theta}{\rho\sin\phi}\dfrac{\partial\rho}{\partial\theta} + \dfrac{\sin\theta\cos\phi}{\rho}\dfrac{\partial\rho}{\partial\phi} \\ \cos\phi - \dfrac{\sin\phi}{\rho}\dfrac{\partial\rho}{\partial\phi} \end{bmatrix}$$

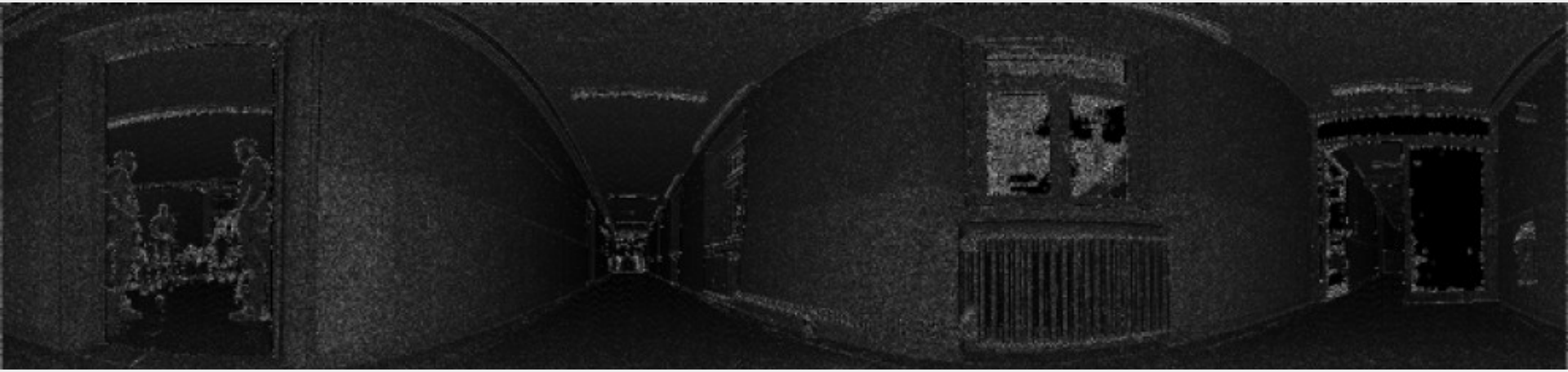
# Results for Panorama Images



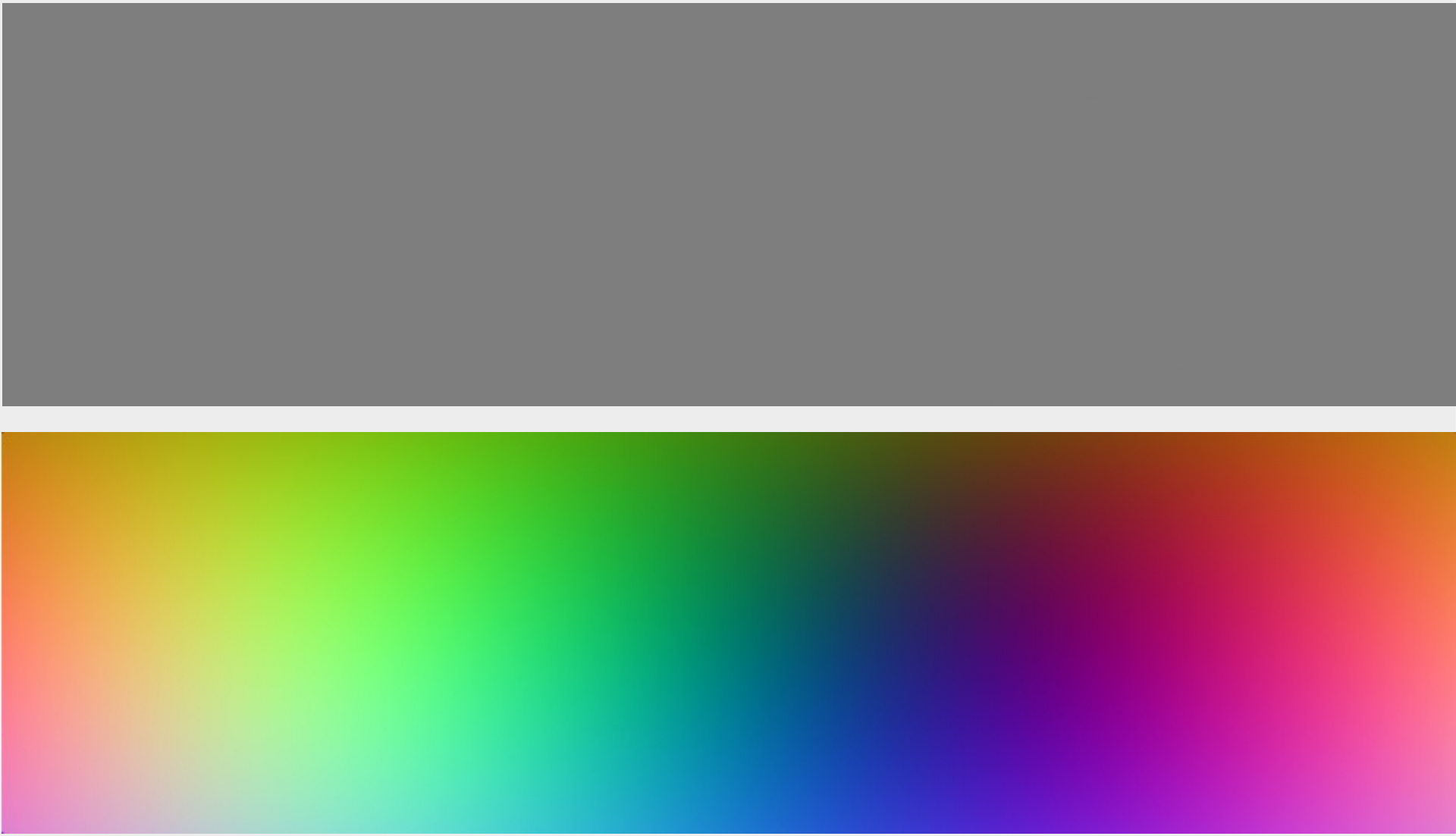


# Results for Panorama Images (2)

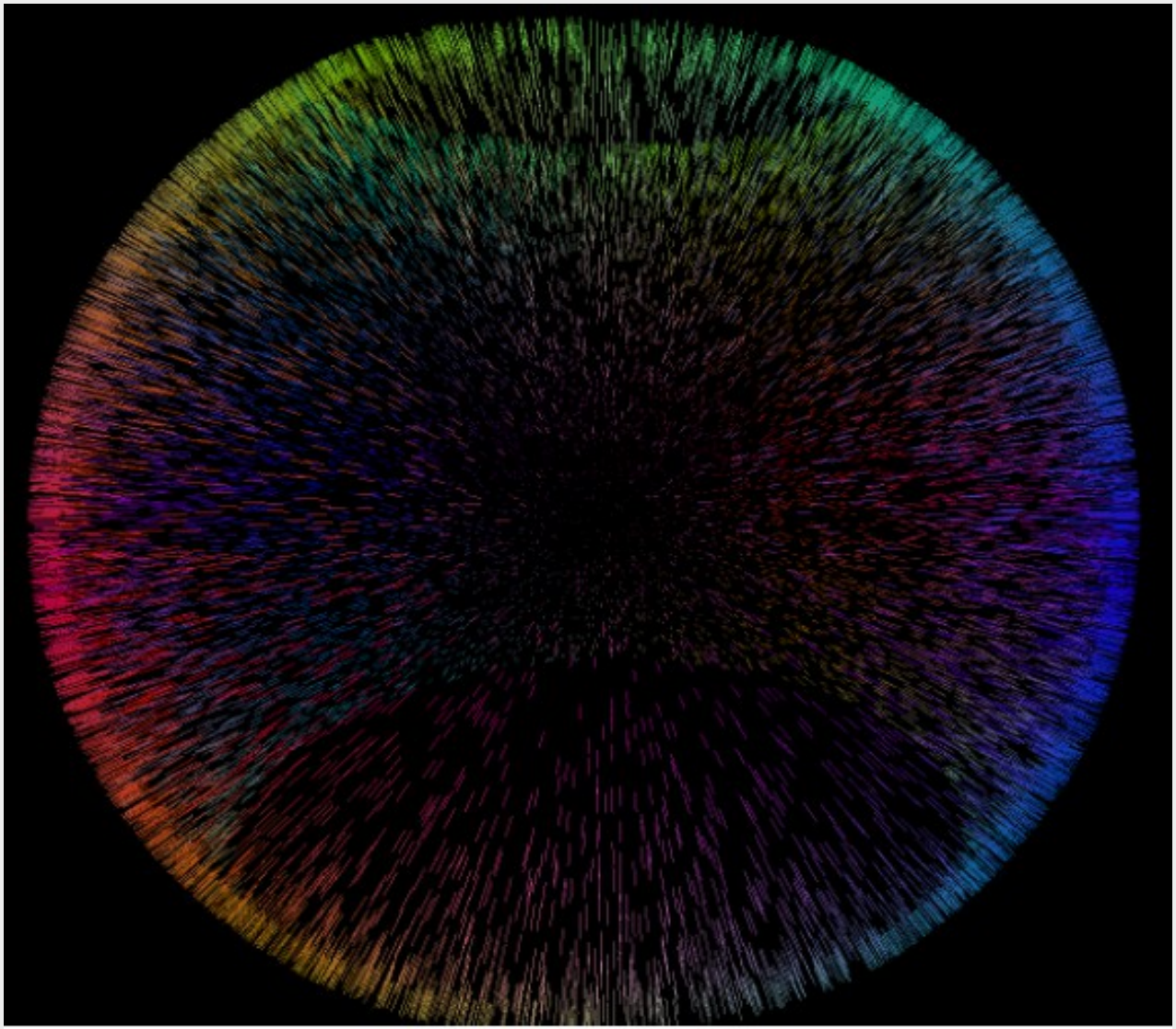




# Results on Synthetic Images (1)



# Results on Synthetic Images (2)



# Calculating Normals

… with *kNN*

**bin/calc_normals -f RIEGL_TXT -g AKNN -k 10**
**-s 0 -e 0 ~/dat/bremen_city**

… replacing *k*NN with the panorama image

**bin/calc_normals -f RIEGL_TXT -g PANORAMA**
**-s 0 -e 0 ~/dat/bremen_city**

… normal computation using the derivatives on the spherical range image

**bin/calc_normals -f RIEGL_TXT -g PANORAMA_FAST**
**-s 0 -e 0 ~/dat/bremen_city**

- Viewing the results

**bin/show -s 0 -e 0 -f uos_rgb ~/dat/bremen_city/**
**--color**