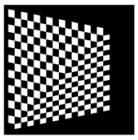
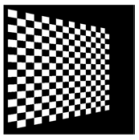
Texture Mapping

Texture Interpolation

Affine Texture Mapping
(Linear Interpolation)



Hyperbolic Interpolation



Perspective-Correct.
Accounts for depth Instead of interpolating *u* and *v* directly, we interpolate:

At each vertex i of a triangle, you have:
- Texture coordinates: (u_i, v_i)
- Homogeneous depth value: w_i (from projection transform)
- Screen-space barycentric weights: $\lambda_1, \lambda_2, \lambda_3$

At each vertex i, compute:

$$u_i' = \frac{u_i}{w_i}, \quad v_i' = \frac{v_i}{w_i}, \quad z_i' = \frac{1}{w_i}$$

Using barycentric interpolation:

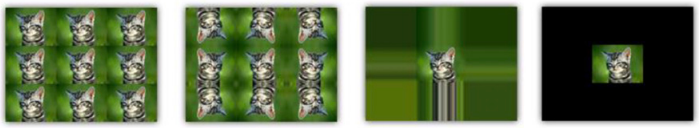
$$u' = \lambda_1 u_1' + \lambda_2 u_2' + \lambda_3 u_3'$$
$$v' = \lambda_1 v_1' + \lambda_2 v_2' + \lambda_3 v_3'$$
$$z' = \lambda_1 z_1' + \lambda_2 z_2' + \lambda_3 z_3'$$

$$u = \frac{u'}{z'} = \frac{\lambda_1 u_1' + \lambda_2 u_2' + \lambda_3 u_3'}{\lambda_1 z_1' + \lambda_2 z_2' + \lambda_3 z_3'} = \frac{\sum_i \lambda_i \frac{u_i}{w_i}}{\sum_i \lambda_i \frac{1}{w_i}}$$

$$v = \frac{v'}{z'} = \frac{\lambda_1 v_1' + \lambda_2 v_2' + \lambda_3 v_3'}{\lambda_1 z_1' + \lambda_2 z_2' + \lambda_3 z_3'} = \frac{\sum_i \lambda_i \frac{v_i}{w_i}}{\sum_i \lambda_i \frac{1}{w_i}}$$

Typically $w = 1$, in clip-space (Homogeneous coordinates, not normalized).

Wrap Modes



GL_REPEAT | GL_MIRRORED_REPEAT | GL_CLAMP_TO_EDGE | GL_CLAMP_TO_BORDER

Texture Filtering (= Interpolation between Textures)

Magnification Filter: Screen pixel maps to an area less than one texel. Texture is upscaled, creating new pixels through interpolation. Maps fewer texels to more screen pixels.

Minification Filter: Screen pixel maps to an area greater than one texel. Texture is downscaled through averaging. This is more critical → **Mipmaps**

Filtering Algorithm: nearest, linear, bilinear
Triliear (Mipmap) Filtering: Interpolating between mipmap texutes.
Anisotropic Filtering: Multiple interpolations between mipmap texutes. Gives better approximation for the average color of that pixel. But expensive.

Mipmaps

MIP: latin. *Multum in Parvo*, Many things in a small place / Much in little

A sequence of textures and of different resolutions. Always half the resolution (4x smaller) of the previous version. For distant objects, lower resolutions are used and then prefiltered.

UV Mapping

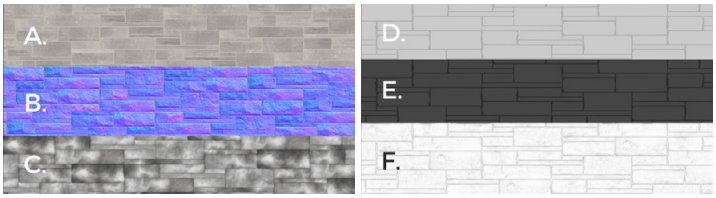
Transferring a mesh onto a 2D texture.

Physically Based Rendering (PBR)

Emulates the interaction between light and materials.

Texture/Map Types:

Albedo / Diffuse (A): Base Color. Texture should contain no shadows.
Normal / Bumb (B): Displacement of normal vectors.
Simulates small surface detail. Fake geometry.
Displacement (C): Height Map that modifies the geometry.
Roughness (D): Controls specular reflections.
Reflection / Metalness (E): Indicate the intensity of the specular reflection: Non-conductive materials will typically reflect the color of the light source. For conductive materials (metals) the reflection will be colored and referred to as the reflectance color.
Ambient Occlusion (F): Simulate subtle shadows.



Refraction and Environment Map:

Effective way to render shiny objects that reflect the environment or to simulate transparent objects. Converts a reflection vector into texture coordinate, which represents the environment.

Texture Types
- Cube Map: Most performant. Supported by OpenGL.
- Sphere Map / Matcap: Simpler but more distortion.

Shadow Mapping

Shadow map algorithm consists of two passes:

1. Render the scene from the point of view of the light and store as a depth map = **Shadow Map**
2. Compare the depth map of the scene with the cameras shadow map. If the depth value of the scene is smaller (= distance is greater) → pixel is in shadow

Computer Animation

Flickering:
- Below **24 fps** (frames per second).
- When displays refresh rate is below **60 Hz** (every 16 ms)

Animation Techniques

Formulas (scripts), Keyframe Interpolation,
Physics simulation, motion capturing

Inbetweens: Frames between Keyframes.

The 12 Principles of Animation

Squash and Stretch: Gives a sense of weight and flexibility.
Timing: Controls speed and rhythm of an action.
Anticipation: Prepares the audience for an action.
Staging: Directs the audience's attention and sets the context.
Follow Through and Overlapping Action: Adds realism by showing how parts of the body or objects continue moving after the main body stops.
Straight Ahead Action and Pose to Pose:
Two different techniques for drawing movement.
- **Straight Ahead:** Animate frame by frame from start to end.
- **Pose to Pose:** Key poses first, then fill in — more controlled and planned.

Slow In and Slow Out: Makes movement feel natural.
Arcs: Natural actions follow a curved path or arc, not straight lines.
Exaggeration: Emphasizes actions or emotions to make them more readable or entertaining.
Secondary Action: Adds depth and nuance to the main action.
Appeal: The character or scene should be engaging and interesting.
Solid Drawing: Giving forms in three-dimensional space volume and weight.

Forward and Inverse Kinematics

Forward Kinematics
Define the movements at each joint. You control each joint manually
You rotate joints from the root outward to compute the final position of the end effector. The final position is computed by applying all those joint transformations (hierarchy of transforms).

Pros:
- Simple and fast.
- Great for predictable motion (e.g., walking, waving).
- Easy to animate with keyframes.

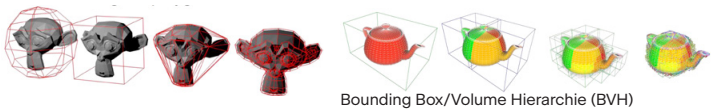
Cons: Hard to position the end effector precisely

Inverse Kinematics
Define the movements at the end points and compute the angles required at each joint. You specify the desired position of the end effector, and the system (IK Solver) calculates how to rotate the joints to reach that point (reverse the chain). Can usually not been solved analytically. Numerical and incremental methods converging to the solution are used (e.g. Jacobian matrix of partial derivatives).

Pros: Great for precise control of hands, feet, or tools.
Works well for interactive animation and physics.
Cons: More complex and computationally expensive. May have multiple or ambiguous solutions. Can look unnatural without constraints (elbow flipping).

Collisions

Optimization by refinement of collision shape:
Bounding Box → Convex Hull → Actuall Polygon Mesh



Bounding Box/Volume Hierarchie (BVH)

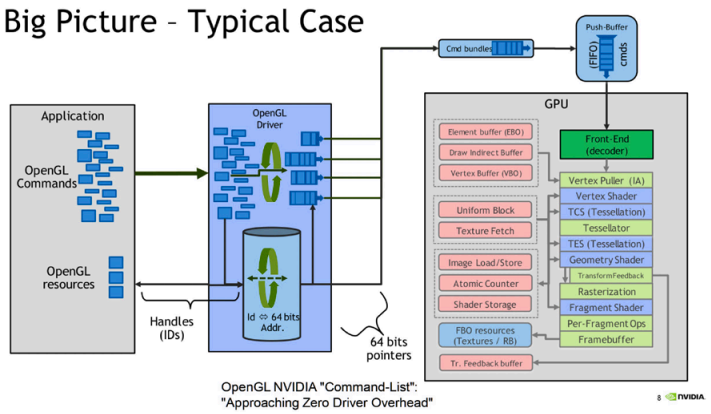
Augmented Reality

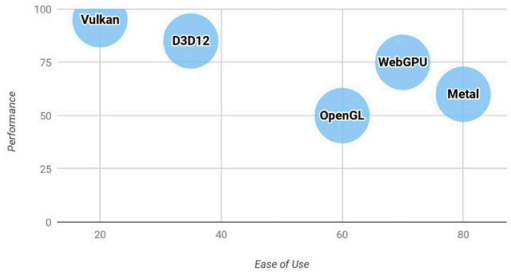
Augmented Reality combines real and virtual objects in a real physical environment and shows both to a user. The user thereby can move around in the real world and the augmentation must occur in real-time.

A combination of virtual and physical objects in the same interaction space by overlaying (superimposed) computer graphics onto the real world.

One of the major problems in interactive computer graphics is how to use a two-dimensional interface with three dimensional objects.

GPU and APIs



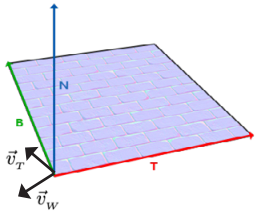


https://fosdem.org/2020/schedule/event/rust_webgpu/

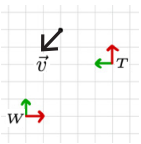
Shader Programming

Tangent Space

Tangent space is a space that's local to the surface of a triangle. Der Tangent Space ist der Raum, in welchem die Texturkoordinaten definiert sind.



$$\vec{v}_W = TBN \cdot \vec{v}_T$$
$$\vec{v}_T = TBN^T \cdot \vec{v}_W$$



The TBN matrix **(Tangent, Bitangent** and **Normal)** is a change-of-basis matrix. It transforms **directional vectors** from tangent space to world space. The TBN cannot be directly used to transform positions, because it is not a rotation matrix.

The inverse (transpose) of TBN is used to transform directional vectors from world space to tangent space. The TBN matrix is orthonormal (basis vectors are orthogonal to each other), so the inverse is equal to the transpose.

Resources: https://learnopengl.com/Advanced-Lighting/Normal-Mapping

Consistent Tangent Space

MikkTSpace (Mikkelsen Tangent Space): Industry standard algorithm for generating smooth, consistent tangent spaces across arbitrary meshes, used for both decoding and encoding normal maps. Propagates tangent orientations across the mesh to avoid discontinuities.

Modelling of the Camera

Recovering 3D

Passive: Passive methods rely on natural light and images, no special sensors or emitters involved.

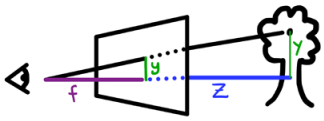
- **Uni-directional:** Only one viewpoint **(a single image)** is used. Depth or 3D structure is inferred using **machine learning.** It makes certain assumptions about the world.
- **Multi-directional: Multiple views** of the same scene from different angles or over time. Examples: Stereo Vision, SfM (Structure from Motion), SLAM

Active: Use of additional sensors (infrared, lasers or projected light) to help measure depth directly.

- **Uni-directional (time of flight):** Measures how long it takes for light to bounce back (e.g. Kinect v2, LiDAR in iPhones).
- **Multi-directional:** Line Scanning, Photometric Stereo

SLAM (Simultaneous Localization and Mapping): Technique used by devices (like robots, drones, or augmented reality systems) to build a map of an unknown environment while simultaneously tracking their own location within it. E.g. Lidar (laser sensors) and depth sensors.

3D to 2D Projection



Triangle Similarity:

$$\frac{x}{X} = \frac{f}{Z}, \quad \frac{y}{Y} = \frac{f}{Z} \qquad x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}$$

f = focal length, (x, y) = image point, (X, Y, Z) = scene points

Camera Calibration and Marker Based Tracking (ArUco)

Radiometric Calibration

The process of correcting and modeling how a camera captures light intensity (radiance) and converts it into pixel values (digital numbers). Includes: **Vignetting, Exposure Correction, hot/dead/defective pixels.**

Correction:

$$\frac{(I - B) \cdot W}{\text{mean}(W)}$$

I: The observed image (typically grayscale)

B: The bias or black level of the camera, i.e., the sensor response when no light is hitting the sensor (used to remove base-level noise).

W: A vignetting map or gain map, which compensates for brightness fall-off from the center to the corners of the image

Geometric Calibration

Such as Radial Distortion due to lens effects.

Homography

youtube.com/watch?v=l_qjO4cM74o&t=23s

2D to 2D Transformation. Very fast but coarse (ungenau), jitters a lot
→ PnP (Perspective n Point)

Marker Based Tracking Pipeline (using Homography)

1. **Camera Calibration:** Calculate Homography Matrix H to get extrinsic camera Parameter.
2. **Marker Detection:** Detect the marker's corners in the image. A rectangular marker provides 4 corners points, which are enough for pose estimation.
3. **Pose Estimation (Position and Rotation):** From 4 point correspondences (marker corners), estimate Homography Matrix H.

Camera Calibration

Camera calibration estimates intrinsic parameters and requires solving for multiple unknowns: **focal length (Brennweite), Optical Center (principal point)** and **distortion coefficients.**

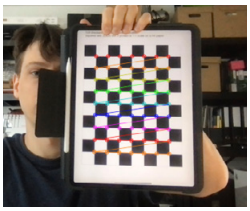
To get a well-posed system of equations, one needs **at least 3 views** to gather enough equations to solve for the intrinsic matrix. Each view (image)

of the plane of a **known Planar Pattern** (2D pattern) gives one **Homography Matrix *H*** = K [R | T], which contributes two (independent) equations (one for x' and one for y'). Man sucht nach **Punktkorrespondenzen** (Corners).

Each homography provides constraints on the intrinsic parameters of the camera. Each H gives 2 constraints. K has 5 intrinsic parameters (typically fx, fy, cx, cy, t).

The Homography Matrix has 8 DOF (Freiheitsgrade). Objekte weit weg können gleich gross wirken wie nahe Objekte. One degree of freedom is lost to this arbitrary scale → Only 8 values matter to define a homography.

Planar Pattern thats needed for the camera calibration:



Camera Calibration Matrix

$$(u,v) \sim K\,[R \mid T]\,P$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} f/s_x & 0 & o_x & 0 \\ 0 & f/s_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & -t_x \\ r_{21} & r_{22} & r_{23} & -t_y \\ r_{31} & r_{32} & r_{33} & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}$$

Extrinsic Camera Parameter: Rotation and translation, 6 DoF
Intrinsic Camera Parameter: Projection and affine transformation (skew and distortion), Focal length, principal point (optical center), 5 DoF
Total 11 DoF

f = focal length, *s* = Pixel ratio, *o* = Principal point (optical center, Urpsung).

Example:

$$\begin{bmatrix} 965 & 0 & 638 \\ 0 & 965 & 366 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

$fx = f/s_x$ and $fy = f/s_y$ are the focal lengths in pixels. s_x and s_y are pixel ratio. c_x and c_y are the principal point (optical center).

Affine Transformation: Geometric transformation that **preserves lines and parallelism.** An affine transformation changes the shape or position of an object using combinations of: **Translation, Scaling, Rotation, Shearing.**

We know that all points lie on a plane in the Planar Pattern (2D pattern): W = 0 and $r_3 = 0$. We are left with:

$$(u,v) \sim K\,[R \mid T]\,P$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} f/s_x & 0 & o_x \\ 0 & f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & -t_x \\ r_{21} & r_{22} & -t_y \\ r_{31} & r_{32} & -t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ 1 \end{bmatrix}$$

$$\mathbf{H}_{3\times3} = \mathbf{K}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix} = \mathbf{K}\begin{bmatrix}\mathbf{R} \mid \mathbf{T}\end{bmatrix}$$

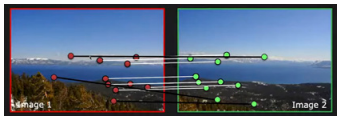
Gesucht ist die Extrinsik-Matrix $[\mathbf{R} \mid \mathbf{T}]$ → DLT-Verfahren

DLT-Algorithmus (Direct Linear Transform)

DLT ist ein Verfahren um u.a. die Projektionsmatrix (Homographie) aus Punktkorrespondenzen zu berechnen. Wird verwendet für Berechnung einer Homographie (2D → 2D), Kamerakalibrierung (3D → 2D Projektion) und Rekonstruktion von 3D-Punkten. **DLT estimates R, T and K (=H)**

DLT (Direct Linear Transform) schätzt die Homographiematrix H zur Berechnung der Homographie (2D → 2D) aus mindestens 4 Punktkorrespondenzen mittels **SVD.**

To solve for *H*, we need:
- at least 4 points per plane
- at least 3 different views of a plane



DLT itself: estimates the **homography matrix** $\mathbf{H}$.
Pose $[\mathbf{R}|\mathbf{T}]$: obtained by decomposing $\mathbf{H}$ with known $\mathbf{K}$. → $\mathbf{K}^{-1}\mathbf{H} = [\mathbf{R} \mid \mathbf{T}]$
→ QR-Verfahren $\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \begin{bmatrix} fx & 0 & o_x \\ 0 & fy & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$

https://www.youtube.com/watch?v=I_qjO4cM74o&t=23s

Combining the equations for all corresponding points:

$$\begin{bmatrix} x_s^{(1)} & y_s^{(1)} & 1 & 0 & 0 & 0 & -x_d^{(1)}x_s^{(1)} & -x_d^{(1)}y_s^{(1)} & -x_d^{(1)} \\ 0 & 0 & 0 & x_s^{(1)} & y_s^{(1)} & 1 & -y_d^{(1)}x_s^{(1)} & -y_d^{(1)}y_s^{(1)} & -y_d^{(1)} \\ \vdots & & & & & & & & \\ x_s^{(n)} & y_s^{(n)} & 1 & 0 & 0 & 0 & -x_d^{(n)}x_s^{(n)} & -x_d^{(n)}y_s^{(n)} & -x_d^{(n)} \\ 0 & 0 & 0 & x_s^{(n)} & y_s^{(n)} & 1 & -y_d^{(n)}x_s^{(n)} & -y_d^{(n)}y_s^{(n)} & -y_d^{(n)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

A (Known) $\quad \mathbf{h}$ (Unknown)

Solve for $\mathbf{h}$: $A\,\mathbf{h} = 0$
such that $\|\mathbf{h}\|^2 = 1$

$A^T A\mathbf{h} = \lambda\mathbf{h}$ Eigenvalue Problem

Perspective n Point

2D–3D point correspondences. Idea: The angle between two lines of sight can be measured with a calibrated camera.

Outside-in Tracking

External sensors determine the user's Position and orientation, e.g. Oculus.

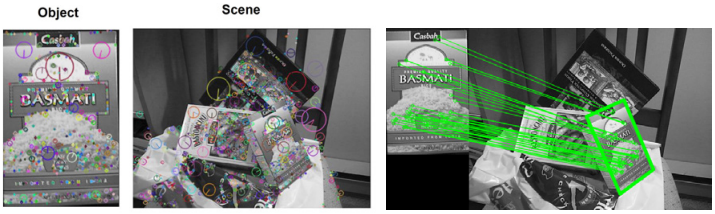
Inside-out Tracking

Visual inertial odometry for pose, e.g. Hololens, Magic Leap

Natural Feature Detection

Object Detection and Object POSE

Finds corresponding image points (interest points).
It describes these interes points with local descriptors (point + surrounding).



Overview

1. Keypoint Detection and Description → SIFT
2. Keypoint/Feature Matching
3. Outlier Removal and Geometric Verification → RANSAC
4. Pose Estimation

1. Keypoint Detection: SIFT (Scale-Invariant Feature Transform)

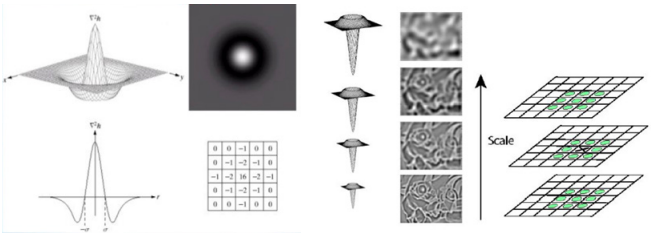
1. youtube.com/watch?v=ram-jbLJjFg, 2. youtube.com/watch?v=IBcsS8_gPzE

1. Scale-invariant: Works at multiple image scales (zoom in/out).
2. Rotation-invariant: Features are robust to image rotation.
3. Partially invariant to affine distortion, illumination changes, and noise.
4. Descriptor generation: compute a 128-dimensional vector that describes gradient magnitudes and orientations in the local region.
   128 = 4x4 cells * 8 histograms (= 8 directions)

1. Scale Invariant (invariant = unabhängig)

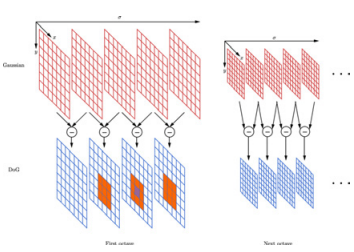
Scale-space Extrema Detection. Finds **keypoints (interest points)** in an image and computes a descriptor for each — a vector that describes the local image structure around that point. Detects local extrema (maxima or minima) by using a small window, e.g. 3x3. Use LoG or DoG.

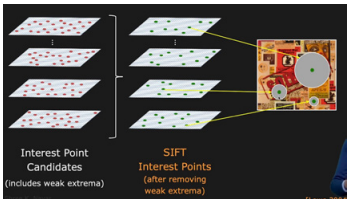
Laplacian of Gaussian (LoG):
Used for **edge detection** with varying line width.



Difference of Gaussians (DoG):

$$\mathrm{DoG}(x,y) = G(x,y,\sigma_1) - G(x,y,\sigma_2) \quad \approx \sigma^2 \cdot \mathrm{LoG}$$





The blob size is proportional to the scale σ.

The extended Difference of Gaussians + Line integral convolution:
Used for stylized images that reduces impact of noise in the image.
https://www.youtube.com/watch?v=5EuYKEvugLU&t=886s

2. Rotation Invariance
Descriptor Computation:

1. Convert to Grayscale
2. Compute **Image Gradients** (using Sobel filters)

$$G_x = I * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, \quad G_y = I * \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Compute gradient **magnitude** and orientation:

$$\text{mag} = \sqrt{G_x^2 + G_y^2}, \quad \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

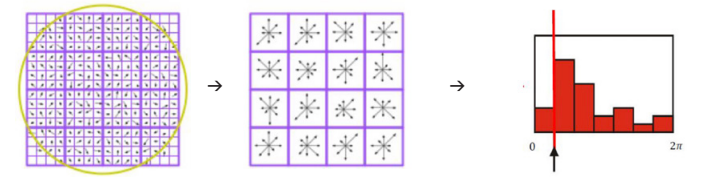
3. Divide Image into patch (cell): Typically 16x16
4. **Compute Keypoint / HOG (Histogram of Oriented Gradients) Descriptor**

Keypoint = location of a locally distinct region in an image

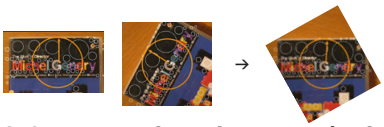
Feature descriptor = vector (the histrogramm) that summarizes the local structure of the keypoint (location)

Create Orientation Histograms: Divide Patch into 4x4 cells
Normalize histograms within blocks to reduce effects of lighting/shadow.

Image Gradients within blob 16x16 patch → **Keypoint Descriptor (HOG)** 128 values (4x4x8) → 1 Histrogram per patch peek = principal orientation



The principal orientationis used to match the rotations in both images:

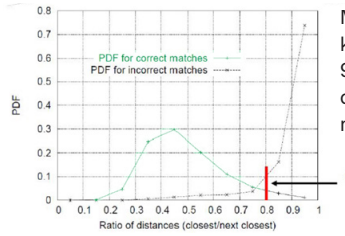


3. Compare and match Features (= Histograms)
Use Normalized Correlation to find matching features (histograms) *H*

$$d(H_1, H_2) = \frac{\sum_k [(H_1(k) - \bar{H}_1)(H_2(k) - \bar{H}_2)]}{\sqrt{\sum_k (H_1(k) - \bar{H}_1)^2} \sqrt{\sum_k (H_2(k) - \bar{H}_2)^2}}$$

where: $\bar{H}_i = \frac{1}{N} \sum_{k=1}^{N} H_i(k)$

4. Get Homography Matrix



Matching a keypoint against obtained keypoints. A threshold of 0.8 eliminates 90% of the incorrect matches while discarding less than 5% of the correct matches.

RANSAC (Random Sampling Consensus)

youtube.com/watch?v=Cu1f6vpEilg, youtube.com/watch?v=EkYXjmiolBg&t=331s

Can cope with a large portion of outliers in the input data. Each point correspondence (e.g. a matched keypoint pair between two images) is a datapoint, that is a pair of matched 2D points from two images (e.g., from SIFT). Idea:

1. Randomly select a pair of corresponding feature points and compute a Homography Matrix H (using DLT).
2. Estimate model (e.g., a homography matrix). Apply H to all other correspondences:
$H \cdot (x, y, 1)^T \approx (x', y', 1)^T$
3. Compare transformed feature point with corresponding feature point and calculate the reprojection error (distance) and count inliers (threshold). Keep the best model estimate with the most inliers.
4. If enough trials then quit otherwise repeat

Choose number of trials *T* so that, with probability *p*, at least one random sample set is free from outliers:

$$T = \frac{\log(1-p)}{\log(1-(1-e)^s)}$$

s Minimum number of samples needed to fit the model
e Outlier ratio

RANSAC is a general algorithm. Example Line-fitting:
Choose two point, a line needs at least 2 points

Dense vs. Sparse

Dense feature descriptor/sampling: Features are extracted uniformly across the entire image for each pixel or often on a grid.
No keypoint detection. Analogy: Like scanning the whole image systematically with a magnifying glass. **HOG Descriptor**

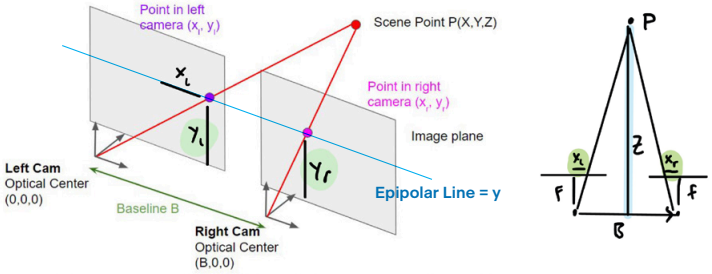
Sparse feature descriptor: Features are extracted only at selected points, typically where the image has strong structure (like corners, blobs, or edges). **Requires a keypoint detector** like Harris, FAST, or Difference-of-Gaussians. Analogy: Like taking photos only at landmarks on a hike — key, meaningful spots. **Keypoint Descriptors (e.g. SIFT, ORB, SURF)**

Stereo Vision and Epipolar Geometry

Stereo

Ziel ist die Tiefe zu schätzen, gesucht ist **Z** in P(X, Y, **Z**)
→ Depth image → 2.5-D Reconstruction

Man benötigt 2+ images und man kennt die horizontale Kamera-Verschiebung (Baseline).



For stereo matching we don't have to search the whole 2D right image for a corresponding point (= match). They all have the same y-coordinate. Thus, one reduces the search space to a line (1D).

$x_r = f\,(X - B) / Z$
$x_l = f\,X / Z$
$y_r = y_l = f\,Y / Z$ (Epipolar Line)

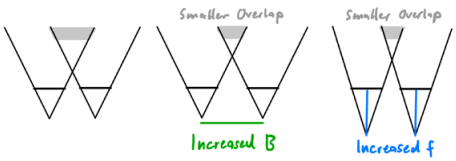
$d = x_l - x_r = f\,X / Z - f\,X / Z + f\,B / Z$
$Z = f\,B / d$

$$Z = \frac{f_{\text{pixels}} \cdot B_{\text{mm}}}{d_{\text{pixels}}} \qquad f_{\text{pixels}} = \frac{f_{\text{mm}}}{\text{Sensor Width (mm)}} \times \text{Image Width (pixels)}$$

Disparity *d* = Unterschied zwischen links und rechts

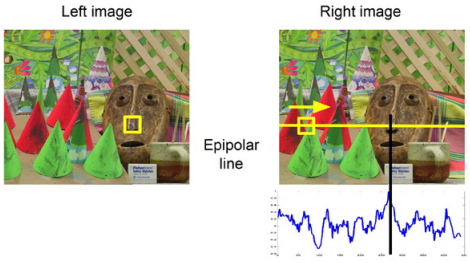
Stereo is **imprecise for far away points**
→ increasing B (baseline) or focal length can increase depth resolution.
→ but overlapping regions are getting smaller.

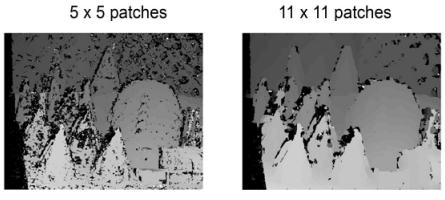
As disparity gets too small, depth difference can no longer be seen.
Human stereo only works up to ~10 m.



Compute Disparity *d*
Idea: Use a sliding window to find best match for x_r. Usually the search range is limited (horizontal pixel shift, called `numDisparities`).

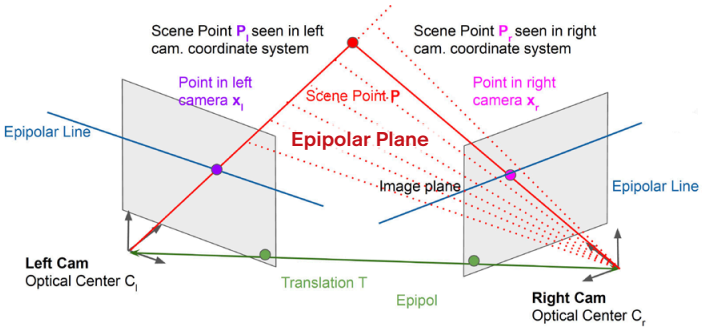
Larger block size implies smoother, though less accurate disparity map. Smaller block size gives more detailed disparity map, but there is higher chance for algorithm to find a wrong correspondence.





Epipolar Geometry

Stereo with Translation and Rotation. Goal is to estimate point **P** (X, Y, Z):
F → E → R, T



Vectors *P* and *T* are coplanar (on the same plane) → Epipolar Plane

Estimating P: F → E → R, T

1. Essential Matrix E

The Essential Matrix E encapsulates the **relative rotation and translation** (i.e., the pose) between two calibrated cameras.

$$E = R\,S \qquad S = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix}$$

R ist the Rotaionmatrix

$$p_r^T\, E\, p_l = 0$$

Meaning: Both points *p* lie in the same **epipolar plane.** Point P must lie on the Epipolar Lines of p_r and p_l and Point *P* projects to p_r and p_l. The line between the two camera centers.

2. Fundamental Matrix F

F captures the relationship between corresponding points in the two views. Describes the epipolar geometry between uncalibrated cameras, in pixel space. It **encodes** both the **relative pose of the cameras and their intrinsic parameters** (focal length, principal point, etc.).

Apply camera model. K are the intrinsic matrices of the left and right cameras.

$$x_l = K_l\, p_l \quad \Rightarrow \quad p_l = K_l^{-1}\, x_l$$
$$x_r = K_r\, p_r \quad \Rightarrow \quad p_r = K_r^{-1}\, x_r \quad \Rightarrow \quad p_r^T = x_r^T K_r^{-T}$$

Plug into essential matrix. It tells you where the match must lie.

$$x_r^T \underbrace{K_r^{-T} E K_l^{-1}}_{F}\, x_l = 0$$

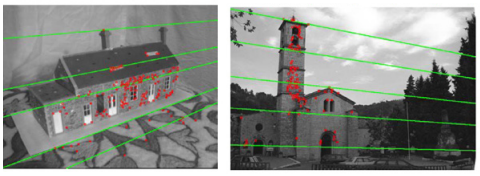
$$x_r^T \quad F \quad x_l = 0$$

Given x_r the epopolar line in the left image is given by

$$u_l = F\, x_r$$

Due to many mismatches, F is often found via **RANSAC.**

Example of Epipolar Lines:



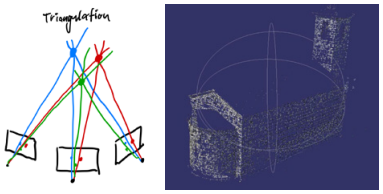
3D Reconstruction Pipeline

Goal: From a sequence of input images, recover 3D scene geometry and camera positions.

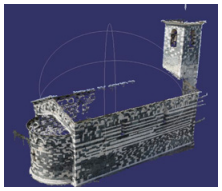
Workflow

1. Local Image Features: **Extract and match Keypoints** and Descriptors in individual views (Get Point Correspondences).
   → Stereo or track keypoints in a video (Sparse Optical Flow, KLT points)

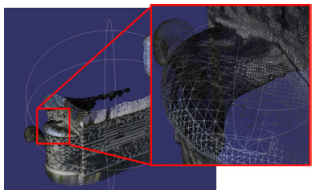
2. Get **Sparse Point Cloud** and Camera Poses
   2.1 Camera Motion: F → E → R, T
   2.2 Scene Geometry: Triangulation (ray intersection)



3. Use Stereo to get **Dense Point Cloud** Reconstruction (Lücken werden aufgefüllt)



4. Recover Surfaces (Delaunay Triangulation)



2.2 Scene Structure Estimation (Triangulation)

$$x_l = K_l\, P \qquad x_r = K_r\, [R \mid T]\, P$$

Only P is unknown. x is known by matching keypoints.
R, T recovered pose with respect to left cam.

The solution seems trivial but in practice there will not be a point that satisfies both constraints because the measurements are usually noisy.

Solution: Best fit → DLT (Direct Linear Transform)
Find the P that lies closest to all of the 3D rays corresponding to the 2D.

SfM vs. SLAM

SfM (Structure from Motion)
Offline (batch) processing
From a sequenze of photo (image data only)
Feature matching
Libraries: COLMAP, GLOMAP, Meshlab, github.com/colmap

SLAM (Simultaneous Localization and Mapping)
Real-time (on the fly)
Dense video
Robotics (odometry data, motion models)
Track camera motion (e.g., KLT tracking)
Intrinsics pre-calibrated

NeRF and Gaussian Splats

NeRF (Neural Radiance Field)

Kinda deprecated, Replaced by Gaussian Splats.

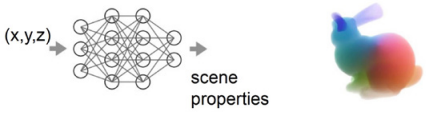
Uses deep learning methods for 3D Reconstruction. Reconstruct geometry by running stereo or multi-view stereo on a set of images.

- Encode convincing view-depth effects using directional dependences
- Encodes detailed scene geometry with occlusion effects
- Encodes detailed scene geometry

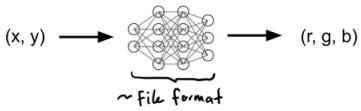
Cannot capture non-Lambertian effects (reflections, transparency, etc.).
Needs markers (Aruco) to calibrate the camera(s).

Volumetric Rendering

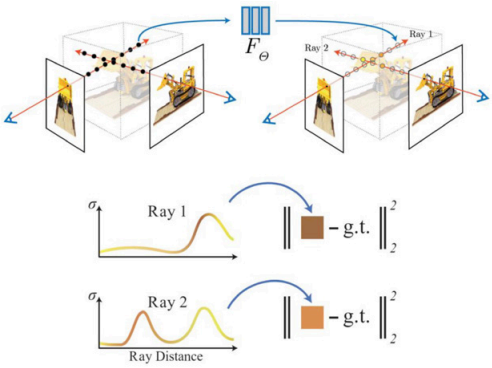
Idea: Query the radiance value along rays through 3D space (volume).
It uses a neural network as scene representation (it stores/encodes a 3D scene in a neural network), rather than a voxel grid of data.



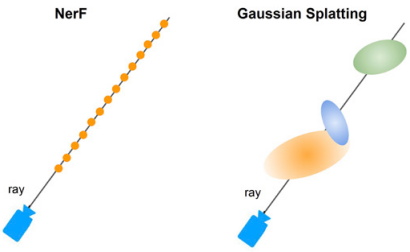
Idea: Example for a 2D image



Integrates color along a ray:

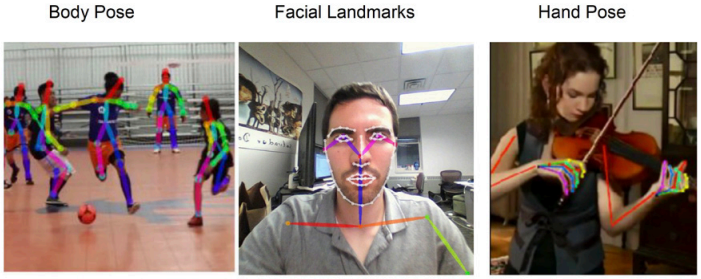


Gaussian Splats



Much faster to render and train than NeRF.
Splats can be manipulated, rays cannot.

Body and Hand Pose Estimation / Detection



Body Part Classification

Keypoint Detection is done via Confidence Maps.

Libraries: MediaPipe (Google), Detectron2, ML Kit (Google), Open MM Lab

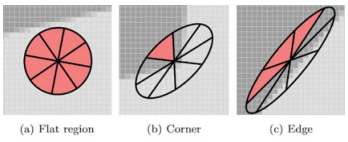
Denoising

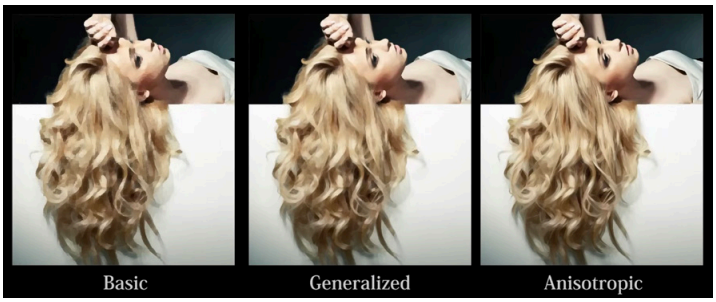
Kuwahara Filter

Originally developed to reduced noise but suffers from boxed shaped artifacts. Now used as a stylized filter.

Better denoise filter: Generalized Kuwahara Filter.

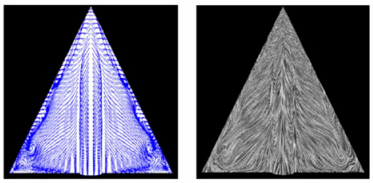
Even better: Anisotropic Kuwahara Filter. Looks very painterly. Takes directions of pixels into account.
https://www.youtube.com/watch?v=LDhN-JK3U9g





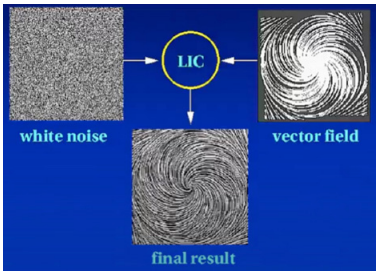
Line Integral Convolution

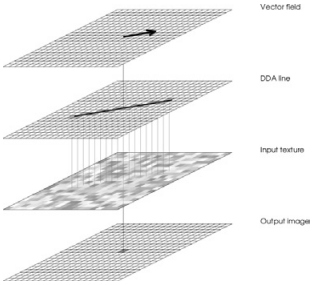
Allows to visualize vector fields. Instead of using line primitives, it will produce a texture (pixel image).



Algorithm (Cabral und Leedom)

Combines a noise texture with a vector field.





To every pixel in the output image:

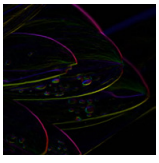
1. Walk along the vector of that pixel (in both directions) = Streamline
2. Take weighted sum of the noise texture along the streamline path.

The weight can be a box filter (pixesl are evenly weighted), tent filter (linearly weigthed by distance) or a 1-d gaussian, or even a windowed hanning rippled function.

Edge Tangent Flow

Sobel filter in x and y direction to determine the direction of the pixel.

atan(Sy / Sx)



Single Value Decomposition (SVD)

Singulärwertzerlegung (SWZ) youtube.com/watch?v=nbBvuuNVfco

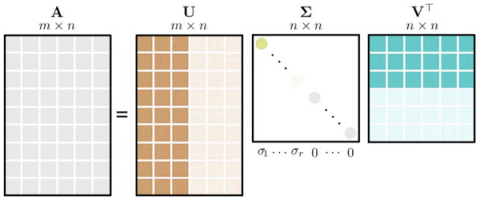
Any Matrix can be decompositioned into 3 Matrices, resp.
any Matrix is the product of 3 Matrices, resp.
SVD can factorize any matrix into a product of three matrices.
SVD also find the direciton of maximum streching.

$$A = U\Sigma V^T$$

$U = AA^T$ Symmetric. Eigenvectors of AA^T

$V = A^TA$ Symmetric. Eigenvectors of A^TA

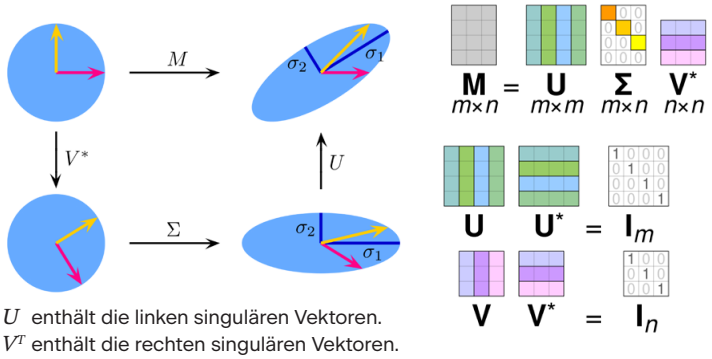
Σ Holds the singular Values σ = sqrt(Eigenvalues)



Die Matrizen AA^T und A^TA **symmetrisch** sind, d.h. die Spalten von AA^T und A^TA Eigenvektoren. Die Eigenvektoren einer symmetrischen Matrix sind parallel (orthogonal) zueinander.

The **singular values** $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$ are the **square roots of the eigenvalues** of $\mathbf{A}^\top\mathbf{A}$ (or $\mathbf{A}\mathbf{A}^\top$).
The singular values σ_i are the **square roots of the eigenvalues** of both $AA^\top$ and $A^\top A$

M entspricht der Transformation:
Rotation/Reflection V^T → Stretch Σ → Rotation/Reflection U



U enthält die linken singulären Vektoren.
V^T enthält die rechten singulären Vektoren.
Σ enthält die singulären Werte.

Kolumnen in U und V^T sind **geordnet.** Erste Kolumnen beschreiben die Informationen in A besser. Die Wichtigkeit ist gegeben durch die singulären Werte σ_1, σ_2, ..., σ_n in Σ. Die singulären Werte in Σ sind ebenfalls geordnet und >0.

$$\Sigma_{N\times M} = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & 0 & \dots & 0 \\ 0 & 0 & 0 & \sigma_4 & \dots & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \dots & \sigma_M \\ 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots \end{pmatrix}$$

$\sigma_1, \dots, \sigma_M$: Singular Values

```
U, s, Vt = np.linalg.svd(A)
S = np.diag(s)
A = U @ S @ Vt
```

Used in: Image compression (see next page), DLT (Direct Linear Transform) Recomender Systems.

Example Matrices:

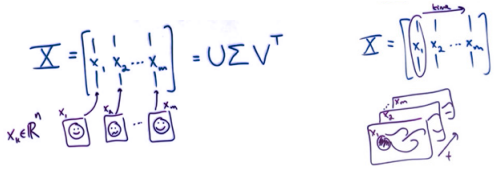
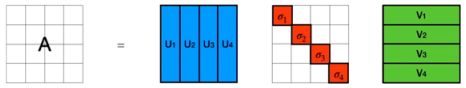
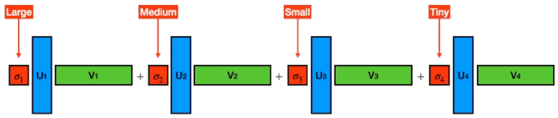


Image Compression:
youtube.com/watch?v=H7qMMudo3e8

Weil kleine Sigmas σ weniger wichtig/beschreibend sind, können diese weggelassen werden.

$$\mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$$



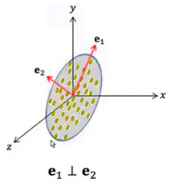


In practice, however, computing the SVD can be too computationally expensive and the resulting compression is typically less storage efficient than a specialized algorithm such as JPEG.

Principal Component Analysis (PCA)

Principal Component Analysisis a **dimentionality reduction** method .

Consider a distribution of points in 3D space that actually lie on a 2D plane. It is redundant to represent each point with 3 coordinates. If we use a new coordinate system {e_1, e_2} that lies on the plane, each point can be represented with just 2 coordniates.



{e_1, ..., e_n} are the **principal components,** ordered by length (=importsance) $e_1 > e_n$. They represent the direction/variance in the distribution. They are **orthonogonal** basis vectors (= orthonormal basis, all parallel to each other, e.g. linearly independent).

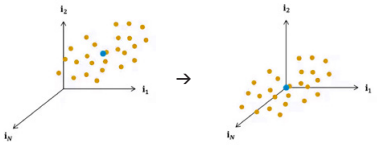
Keyconcept

1. **Start with your data matrix** $X \in \mathbb{R}^{n \times d}$, usually centered (mean subtracted).
2. Compute the **covariance matrix**:

$$\Sigma = \frac{1}{n-1} X^T X$$

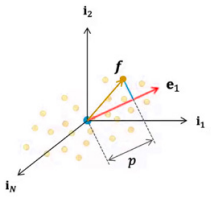
3. Find **eigenvalues** λ_i and **eigenvectors** v_i of Σ such that: $\Sigma v_i = \lambda_i v_i$
   The eigenvectors v_i are the **principal components**.

1. Calculate and substract the mean



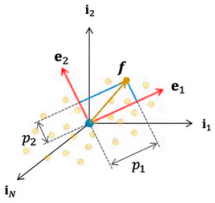
2. The first principal component is equivalent to Least Square Fitting of a line and describes the maximum variance. All point *f* are then projected onto e_1., e.g. all point can be described by e_1.

   Eigenvalue Problem A·e = x·e → Use PCA to solve:
   The columns of *V* are the principal directions (eigenvectors).



$$p = \mathbf{e}_1 \cdot \boldsymbol{f}$$

3. Calculate next principal components



$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = [\mathbf{e}_1 \quad \mathbf{e}_2]^T \boldsymbol{f}$$

Forward Projection
Reduced data from a higher to a lower dimensional space.

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_K \end{bmatrix} = [\mathbf{e}_1 \quad \mathbf{e}_2 \quad \dots \quad \mathbf{e}_K]^T \boldsymbol{f}$$

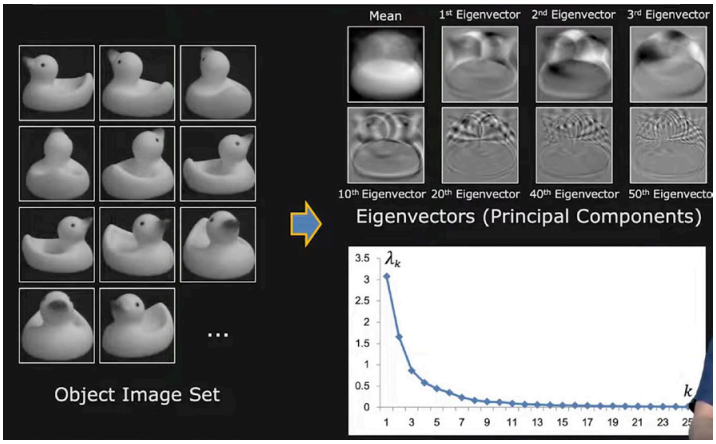
Back Projection
Will give you an approximation of a data point

$$\boldsymbol{f} \approx p_1 \mathbf{e}_1 + p_2 \mathbf{e}_2 + \cdots + p_K \mathbf{e}_K$$

$$\boxed{\boldsymbol{f} \approx \sum_{k=1}^{K} p_k \mathbf{e}_k}$$

Vector *p* and *e* can be used for image **compression.**

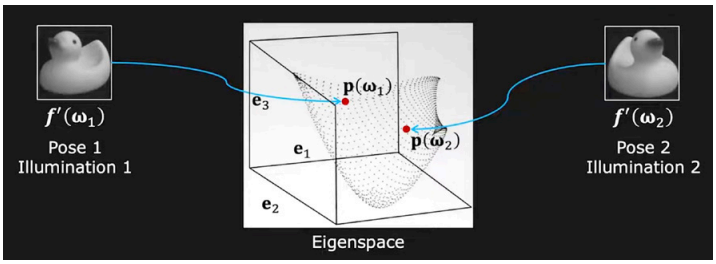
Beispiel: https://www.youtube.com/watch?v=58MDlj5xS1s



How many principal components (K) are sufficient if we want to capture 95% of variation in the data set? Find smalleest K such that:

$$\frac{Sum\ of\ K\ largest\ Eigenvalues}{Sum\ of\ all\ Eigenvalues} = \frac{\sum_i^K \lambda_i}{\sum_j^N \lambda_j} \geq 0.95$$

Example: Dataset (e.g. images) projected onto its Eigenspace.



The idea is to interpolate the values inbetween the points in the dataset as a continuous representation.

Feature Detection (Nachtrag)

https://www.youtube.com/watch?v=4AvTMVD9ig0
https://www.youtube.com/watch?v=nGya59Je4Bs

- **Keypoint = location** of a locally distinct region in an image
- **Feature descriptor = vector** (the histrogramm) that summarizes the local structure of the keypoint (location)

Keypoints: Finding distinct points
Harris corner, Shi-Tomasi corner detection,
Difference of Gaussians

Features: Describing a Keypoint
SIFT: Scale Invariant Feature Transform

SIFT (Scale Invariant Feature Transform)

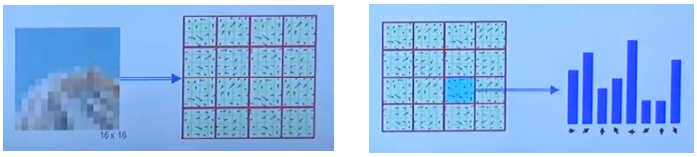
For detecting visuals landmarks from different angels and distances.

Invariant to rotation, rotation and scale. Partially invariant to illuminatin changes and affine transformations and 3D projections.

For a given keypoint, wrap the image around it to orientation and scale and resize the image to 16x16 patch:

Compute thegradients for each pixel (orientation and magnitude) and divide the patch into 4x4 squares.

For each square, compute gradient direction histogram over 8 directions:



Concatenate the histpgrams to obtain a 128 (16*8) dimensional feature vector:

SIFT is expensive to compute. Alternative: Binary Descriptors (binary string) that are easy to compute.

Binary Descriptor

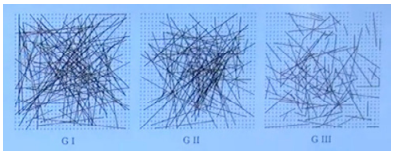
Binary feature detection method. Idea:

1. Select a patch (e.g. 16x16) around a keypoint
2. Select a set of pairs of pixels in that patch
3. For each pair, compare the intensities: $b = 1$ if $I(p_1) < I(p_2)$ else 0
4. Concatenate all b's to a string, e.g. 128 bits (00110100...)

Use Hamming Distance (number of diferent bits) to compare two descriptors:

$$d_{\text{Hamming}}(B_1, B_2) = \text{sum}(\text{xor}(B_1, B_2))$$

Strategies for selecting pairs:
GI: Uniform Random sampling
GII: Gaussian Sampling
GIII: p_1 Gaussian, p_2 Gaussian centered around p_1



→ ORB (Oriented FAST and Rotated BRIEF): Accounts for rotation

https://www.youtube.com/watch?v=0sPlnrEMyYk