Bauhaus-Universität Weimar

Depth Cameras, Demystified

Tutorial at the
2018 Summer School on

Multimodal Interaction
in Augmented and Virtual Reality

by Jun.-Prof. Dr. Florian Echtler
<florian.echtler@uni-weimar.de>

About me

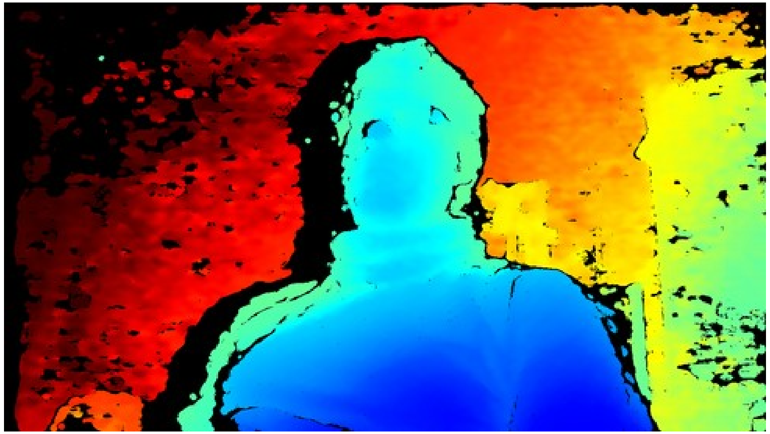
- @floemuc & https://floe.butterbrot.org/
- „Postdoctoral Hacker and Research Hobbyist“
- Co-Author of `libfreenect{2}` open-source drivers for Kinect 1 & 2 depth cameras
- https://github.com/OpenKinect/libfreenect2
- https://doi.org/10.5281/zenodo.50641

Plan for today

- 09:00 ~ 10:00 – Tutoral: depth camera basics
- 10:00 ~ 10:15 – short break
- 10:15 ~ 11:30 – Hands-on depth cam hacking
  (Build Your Own RANSAC)

What are depth cameras?

- „Regular“ camera: color values for each pixel
- Depth camera: distance values for each pixel (usually visualized with color map)



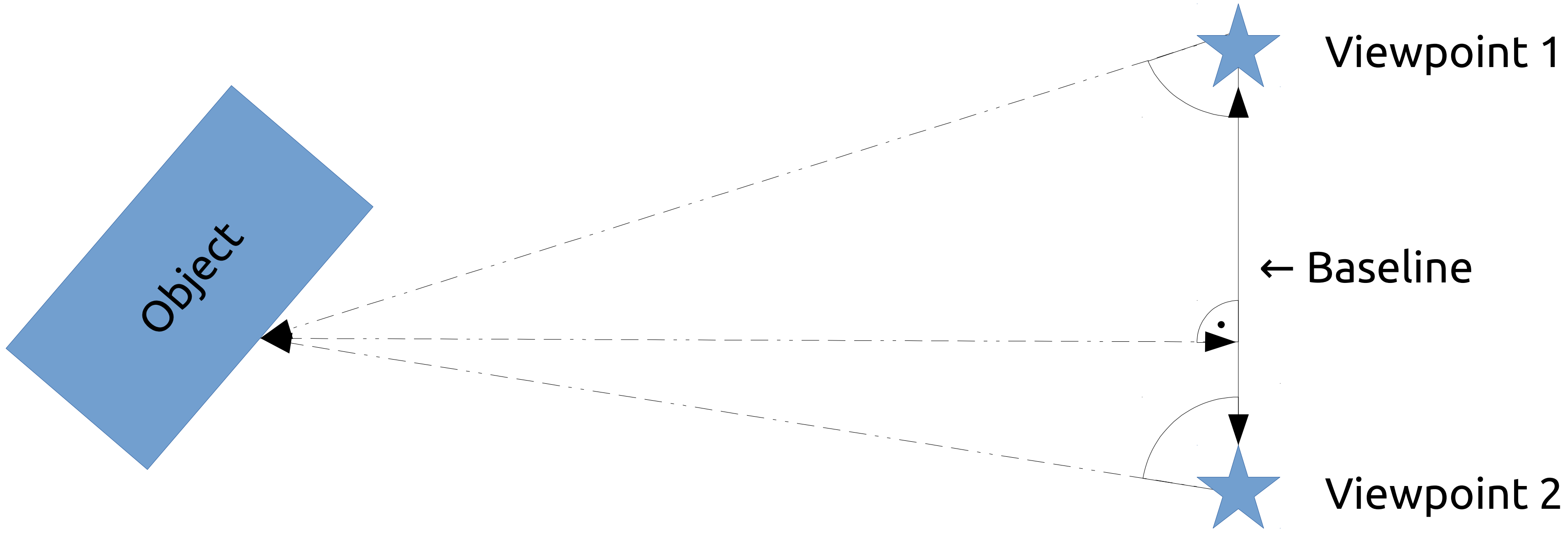
Classes of DCs

- Geometry-based
  - Stereo vision
  - Structured light
- Time-of-Flight/ToF

Geometry-based DCs

- Fundamental principle:
  - Create two „views“ of scene
  - Match scene points between views
  - Determine 2 angles for each scene point
  - Trigonometry happens
  - Receive distance

Geometry-based DCs



Geometry: Stereo Matching

- Method 1: two images of scene

    → *stereo matching* of corresponding pixels

    → ideally only needed on horizontal *scanline*

    – Examples: Occipital Structure Sensor, Intel Realsense D4xx

Geometry: Stereo Matching

Image Source (FU): https://www.mtbs3d.com/phpbb/viewtopic.php?f=138&t=18055

Geometry: Stereo Matching

- Advantages:
  - Close to human vision system
  - Can work outdoors, even in sunlight
  - Only uses plain cameras (* with sync)

Geometry: Stereo Matching

- Drawbacks:
  - Problems with featureless areas (white wall)
  - Mitigation: additional IR pattern projector

Geometry-based DCs

- Method 2: one image of scene + known lighting
  → identify pixel correspondences via IR pattern
  - Replaces one camera with light source
  - 2 sub-methods (down the rabbit hole… :-)

Geometry: Speckle Pattern

- Method 2.1: *speckle* pattern = random dots
  - Random, but previously known pattern
  - „Patches“ of pattern can be matched → depth res. lower than image res.
  - Example: Kinect v1

Geometry: Stripe Pattern

- Method 2.2: Gray code = alternating stripes
  - Encodes binary ID for each pixel
  - Requires high frame rate or static scene (why?)
  - Requires IR projector
  - Example: Realsense SR300 (~ 300 FPS)

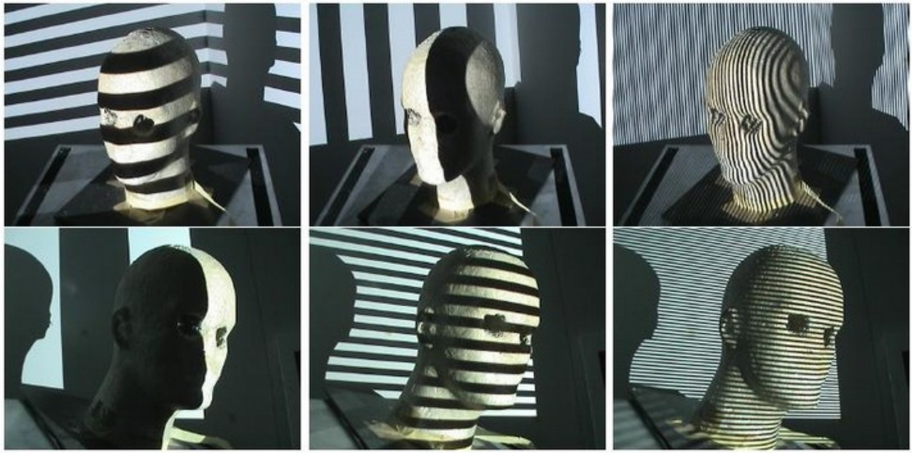
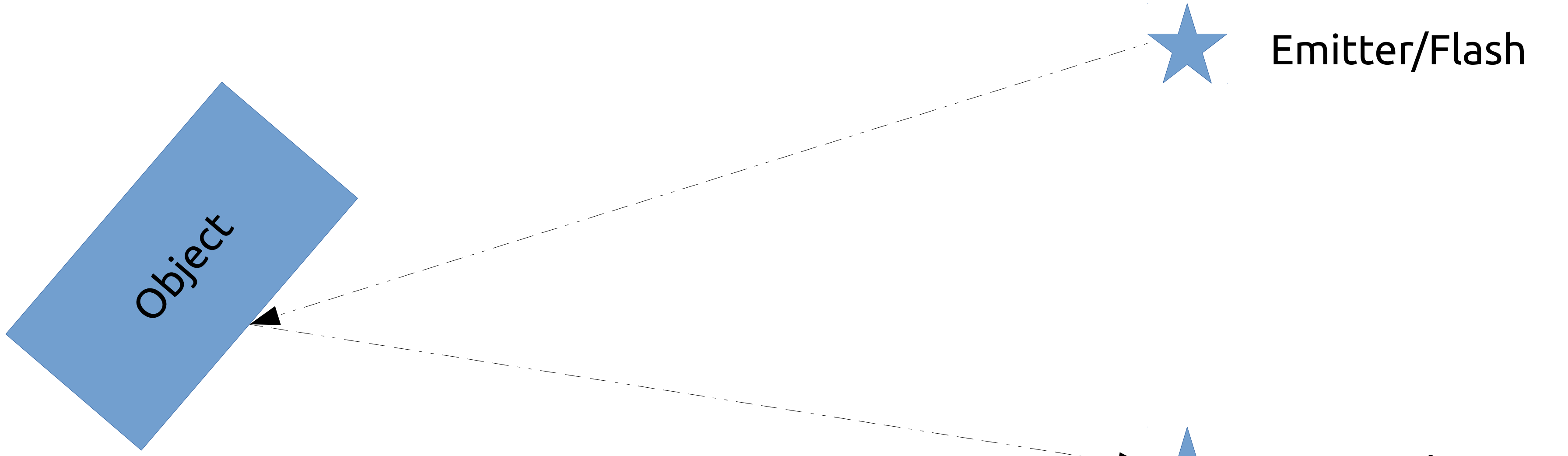


Image Source (FU): http://www.sci.utah.edu/~gerig/.../CS6320-CV-S2012-StructuredLight-II.pdf

Time-of-Flight/ToF DCs

- Fundamental principle:
  - Emit (infrared) flash
  - Measure time until reflected light arrives
  - (very simple) Math happens
  - Receive distance
- Example: Kinect v2

Time-of-Flight DCs



Time-of-Flight DCs

- Problem: c ~ 300 000 000 m/s :-O (~ 671 mil. mph ;-)
- Common ToF depth resolution ~ 1 mm

→ time difference ~ 3 x 10^{-12} s *(3 picoseconds!)*

→ would require counter at ~ 300 GHz per pixel

HOW?

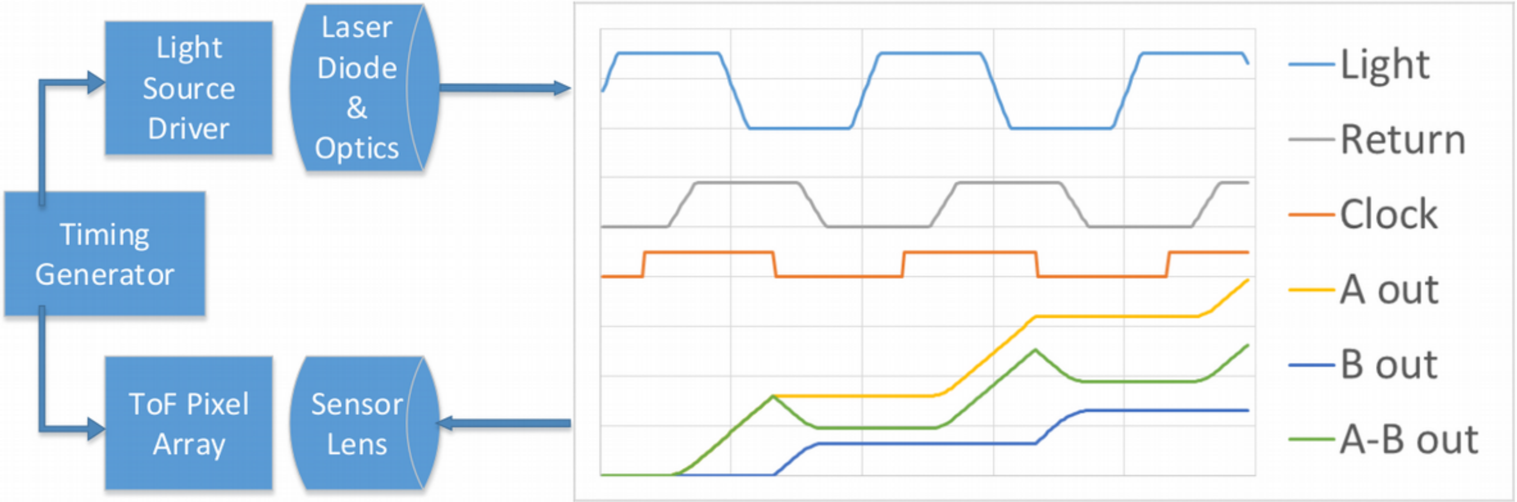
Time-of-Flight DCs

- Solution: measure *phase* difference (not time)
  - Create clock signal with freq. f which modulates ...
    - a) IR emitter („flash“)
    - b) sensitivity of light sensor
  - Result is phase difference
  - Calculation with c and f gives distance

Time-of-Flight DCs

- Requires specialized image sensor
  - Two “accumulators” per pixel (not just one)
  - Clock input switches between acc. A and B
- Kinect v2: result is three measurements per pixel at phase shift 0°, 120° and 240°

Time-of-Flight DCs



Differential Pixel

- (A+B) gives the ambient (room) lighting ('common mode') – 'normal' grey scale image
- (A-B) gives phase (depth) information after an arctan calculation – depth image
- $\sqrt{\Sigma(A-B)^2}$ is the 'Active' image – A grey scale image independent of ambient lighting

Image Source (FU): https://www.hotchips.org/wp-content/uploads/hc_archives/hc25/HC25.10-SoC1-epub/HC25.26.121-fixed-%20XB1%2020130826gnn.pdf

Time-of-Flight DCs

- Tradeoff between precision and range
  - High frequency → better precision, but quicker “wraparound”
  - E.g. Kinect v2 @ 80 MHz: wavelength = 3.75 m
  - Range from 0 to 3.75 m maps to depth resolution of sensor (11 bit for Kv2)

Time-of-Flight DCs

- Solution: switch between multiple frequencies
  - Kinect v2: 16 MHz, 80 MHz, 120 MHz
  - Resolve ambiguity using second/third freq.

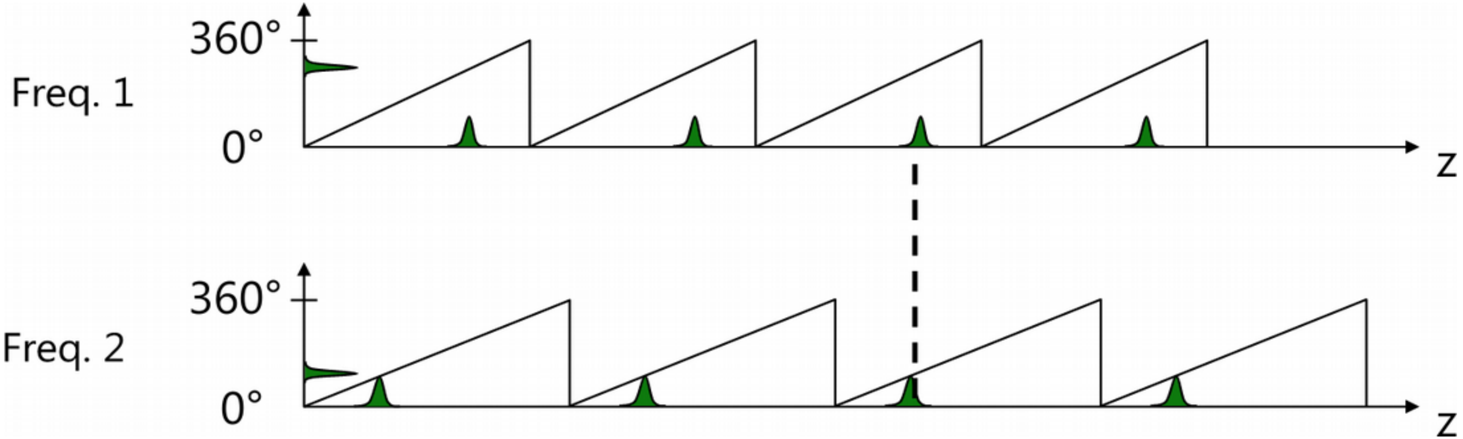


Image Source (FU): https://www.hotchips.org/wp-content/uploads/hc_archives/hc25/HC25.10-SoC1-epub/HC25.26.121-fixed-%20XB1%2020130826gnn.pdf

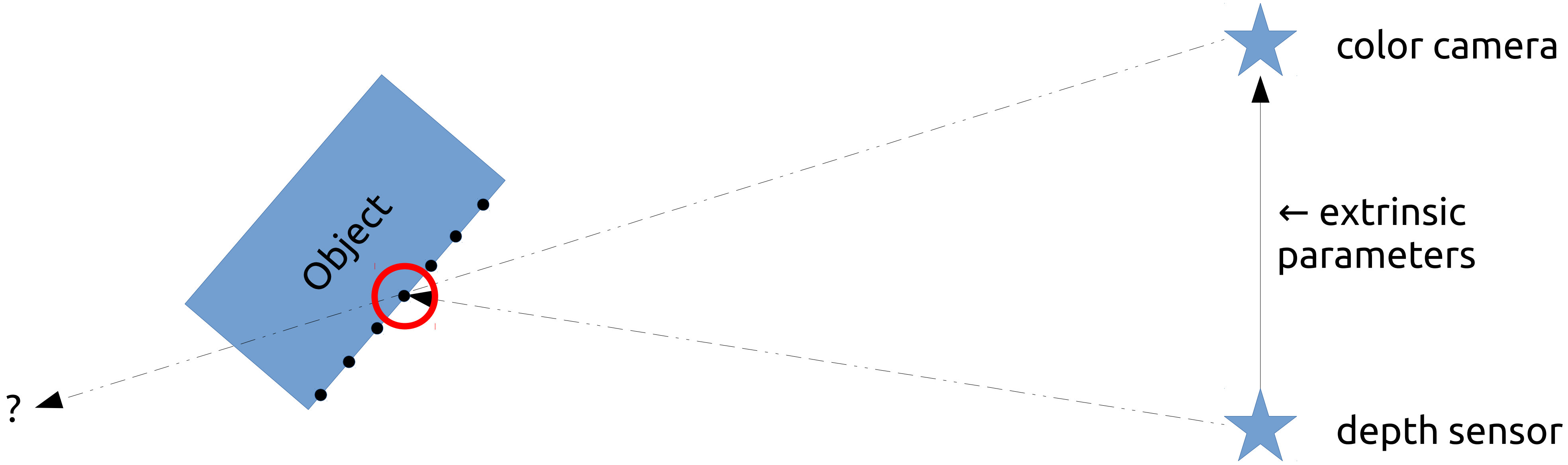
Depth-Color Alignment

- most depth cameras also have a “plain” color cam
- Problem: how to find corresponding color value for each depth pixel (or vice versa)?
- Requires intrinsic and extrinsic camera parameters (?)
  - Intrinsic: field of view, distortion, focal length, …
  - Extrinsic: translation, rotation w.r.t. origin

Depth-Color Alignment

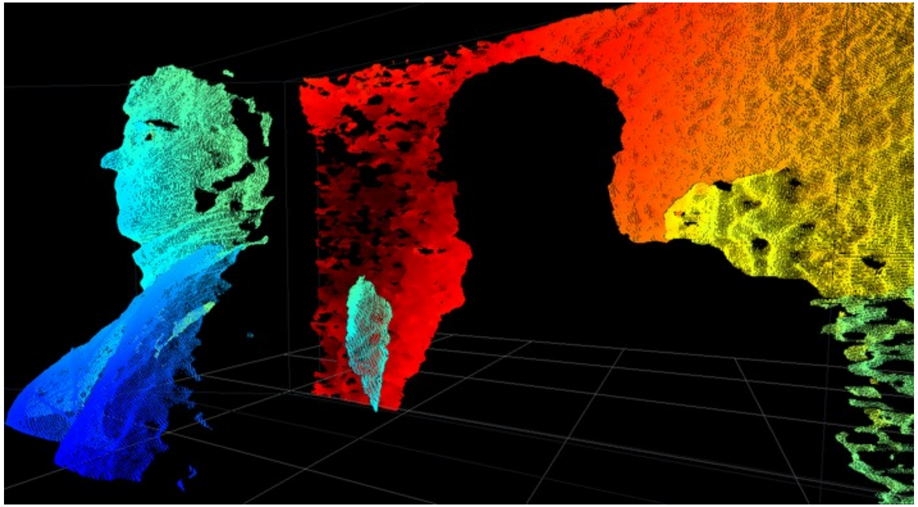
- Alignment process:
  - “Deproject” depth pixels into metric space (needs *intrinsic* parameters of *depth cam*)
  - Translate depth pixels to color camera origin (needs *extrinsic* parameters of *depth cam*)
  - Cast ray in metric space through color pixel (needs *intrinsic* parameters of *color cam*)
  - Find (closest) intersection

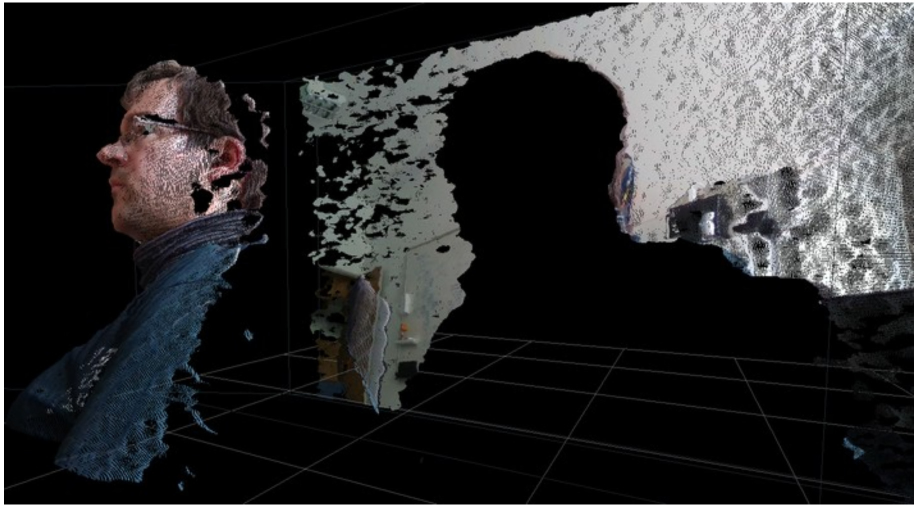
Depth-Color Alignment



Point Clouds

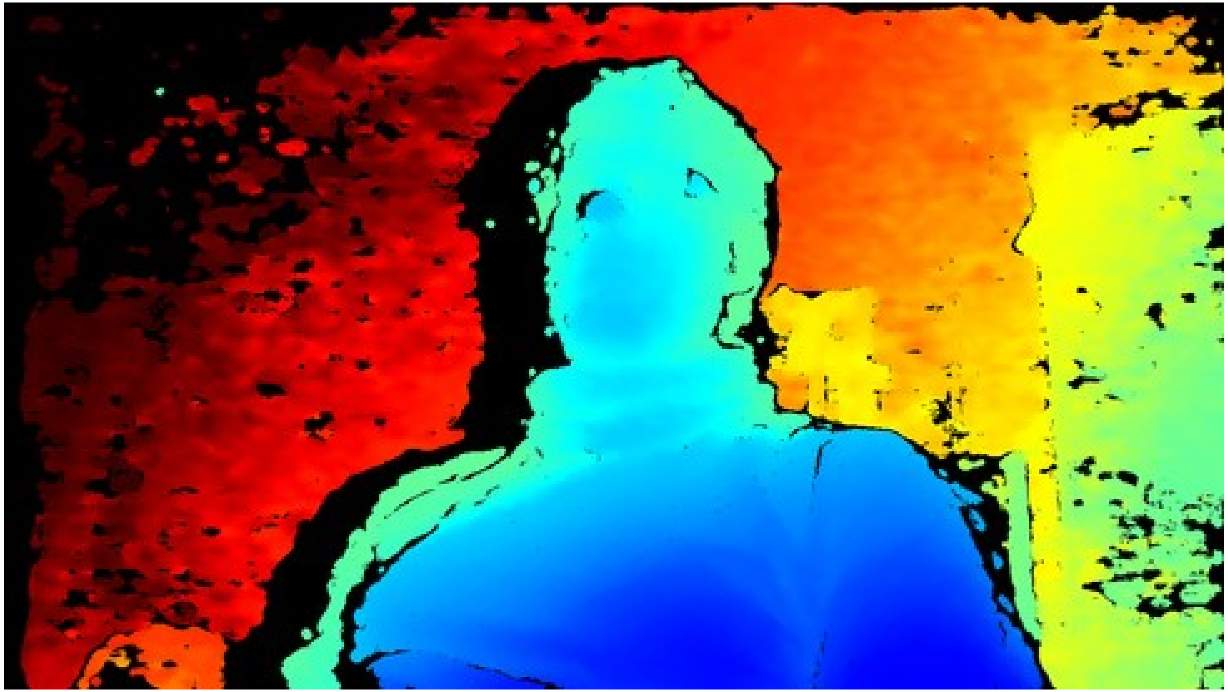
- Each depth pixel → 3D point (x, y, z)
- Plus color → (x, y, z, r, g, b)





Question

- Why the shadow on the left side of the body?



Summary

- One size does not fit all (as always)
  - Lighting? (e.g. sunlight – very hard)
  - Texture? (e.g. flat featureless walls)
  - Absorption? (e.g. some fabrics)
  - Reflections? (nearly impossible ATM)

RANSAC

- RANSAC = RANdom SAmple Consensus
  - Goal: fit a model to noisy data
  - Model: e.g. plane, line, …
  - Data: e.g. point cloud

RANSAC

- Step 1: randomly choose minimum # of data points required for model (e.g. 2 for line)
- Step 2: count data points which are *inliers* (i.e. "close enough" to the model)
- Too few inliers → step 1
- Otherwise: refine model using inliers (optional)

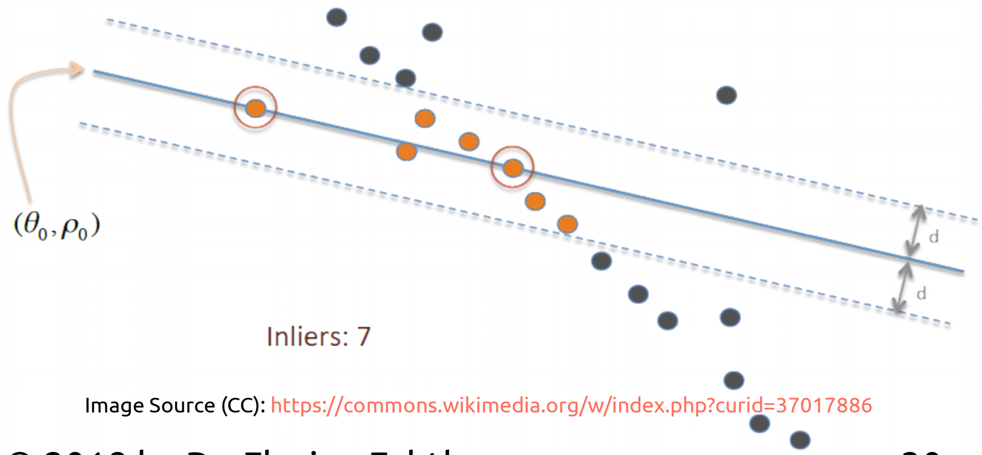


Image Source (CC): https://commons.wikimedia.org/w/index.php?curid=37017886

Demo Time!

- See https://github.com/floe/surface-streams
- and https://www.youtube.com/watch?v=Qe1BROtGyzI

Your turn!

- Pick a depth camera from the pile
- Install the respective SDK
- Adapt the sample code to measure average deviation from plane
- Find a plane
- Profit!