

Learned 3D and 4D Reconstruction

From Visual SLAM to Neural Fields and Gaussian Splatting

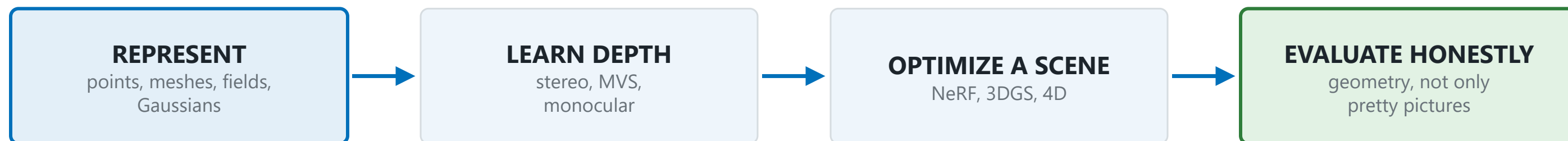
Matouš Cejnek

Department of Instrumentation and Control Engineering
Faculty of Mechanical Engineering, CTU in Prague

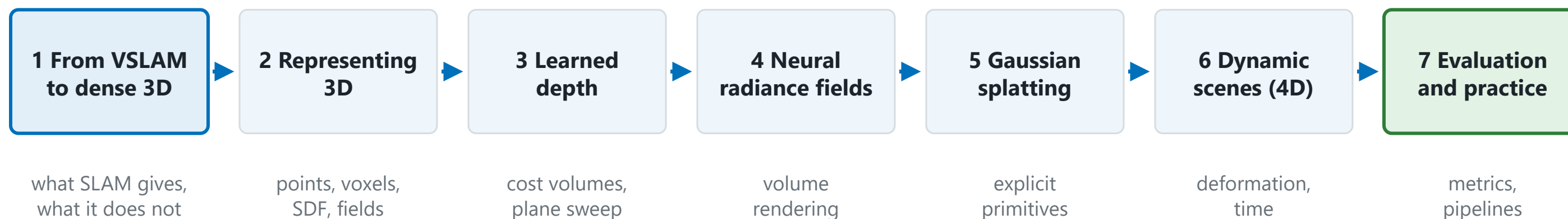
Why This Lecture

Part 1 · From VSLAM to dense 3D

- Visual SLAM gave us camera poses and a sparse map — enough to localize, not enough to measure or render.
- Learned methods fill in dense geometry and appearance, using the CNNs from lectures 1–2.
- They also fail in new ways; we must learn to tell a pretty picture from a correct model.



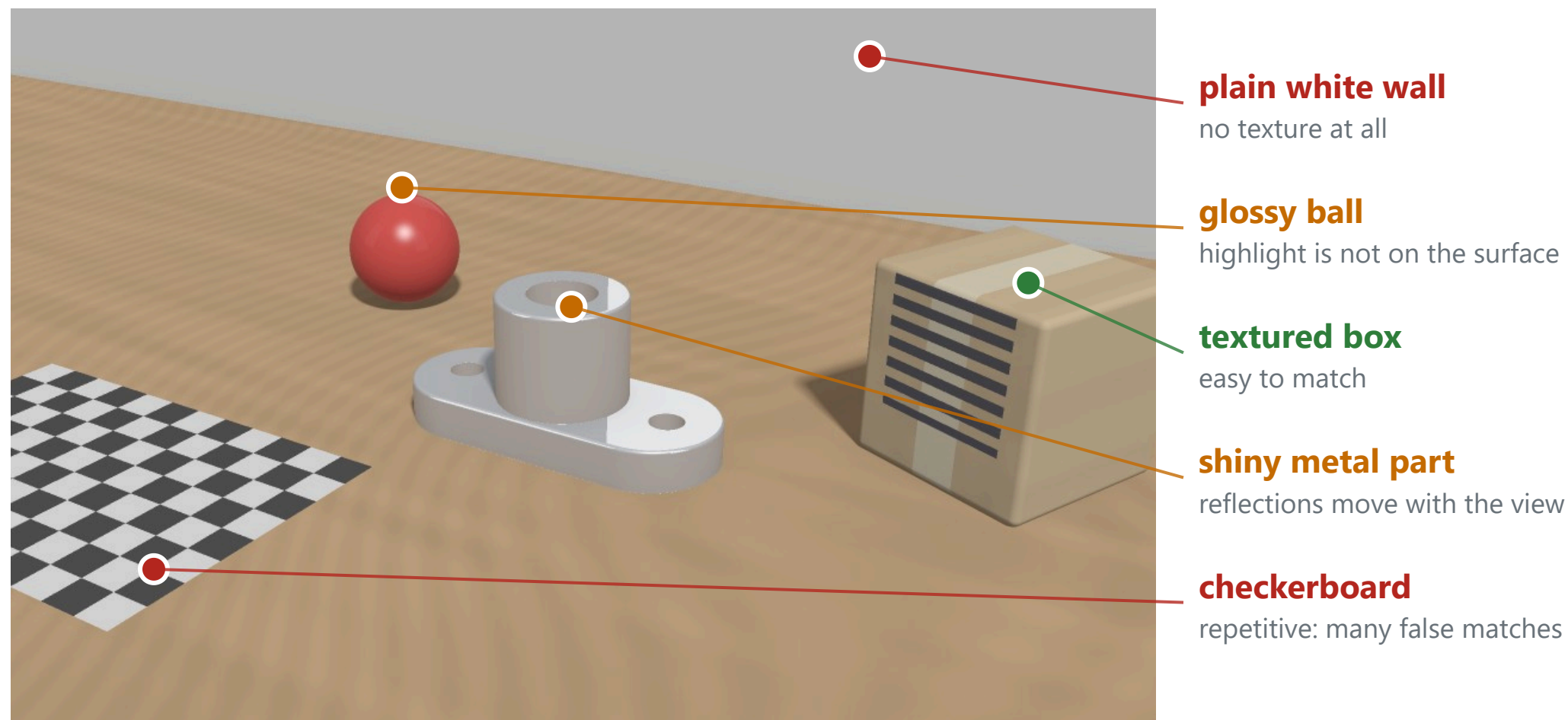
- We start from what the VSLAM block already provides.
- Then representations, learned depth, NeRF, Gaussian splatting, and time.
- The last part is about evaluating results honestly.



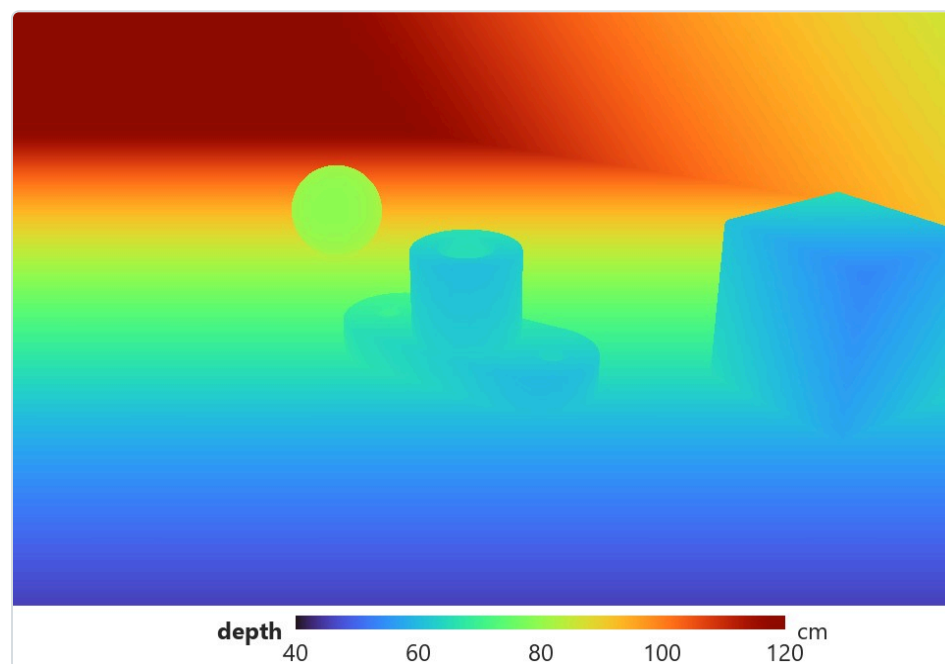
The Scene We Will Use

Part 1 · From VSLAM to dense 3D

- One small workshop scene, rendered with exact geometry — so every error can be measured in millimetres.
- It contains the typical troublemakers of real inspection scenes.
- Every picture in this lecture is computed from it by real algorithms, not drawn.

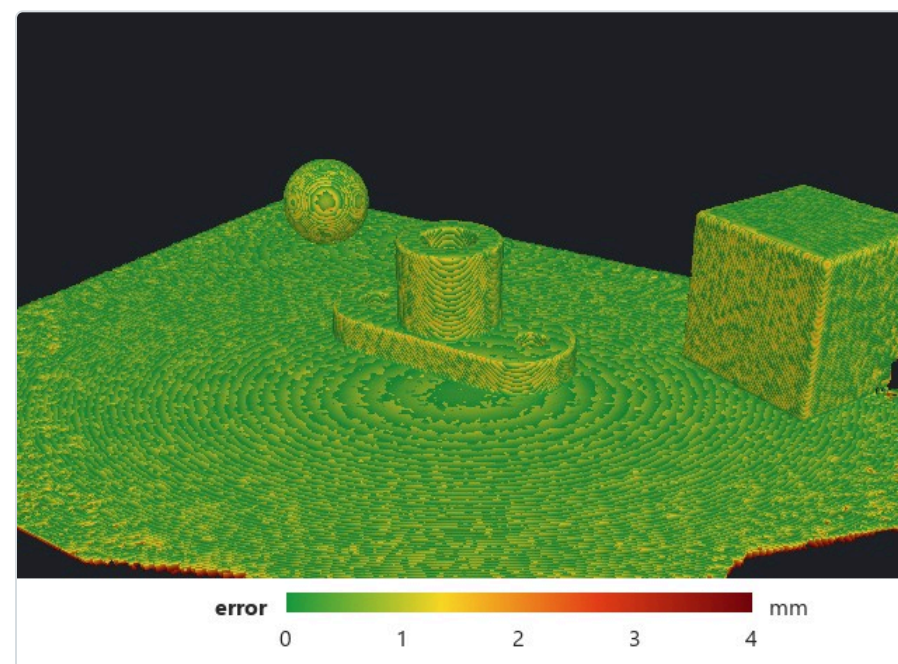


- Where this lecture leads: three kinds of results from photographs.
- Keep these pictures in mind — every part explains one of them.
- The question is always *why* a method is used, and what it cannot do.



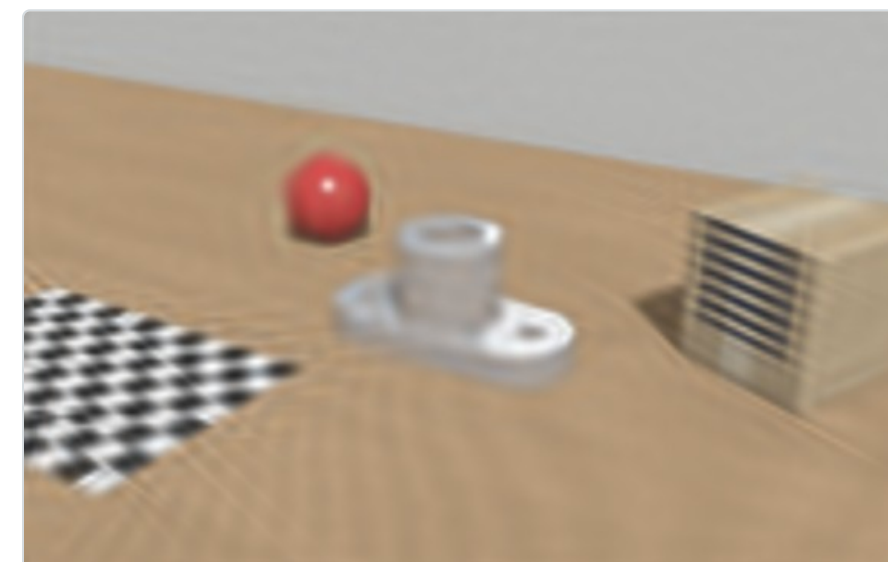
dense depth

a distance for every pixel



a measured surface

coloured by its error in mm



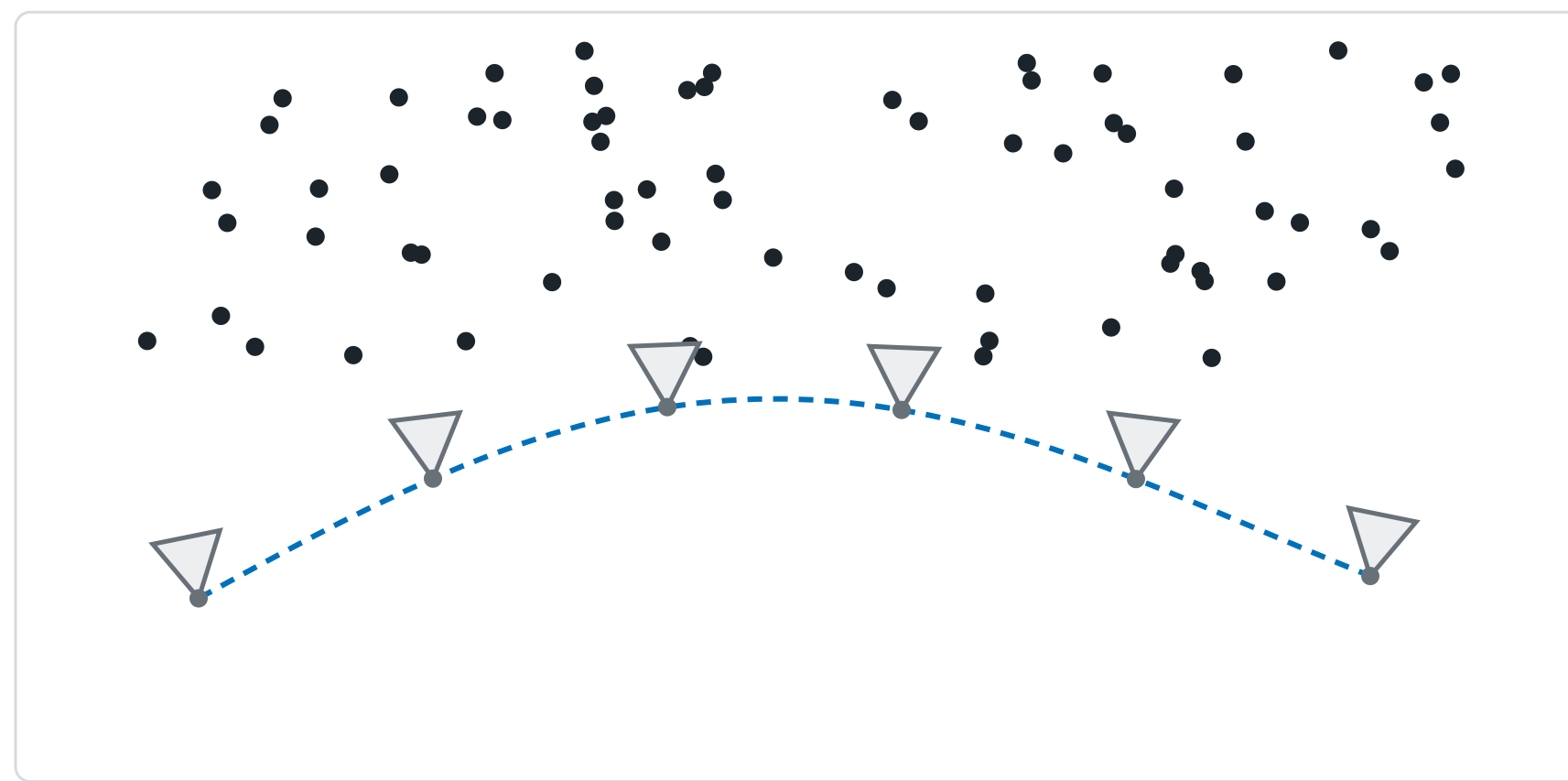
an image rebuilt from 1200 Gaussians

the idea behind Gaussian splatting

What VSLAM Already Gives Us

Part 1 · From VSLAM to dense 3D

- A trajectory of camera poses and keyframes in one consistent frame.
- Sparse 3D landmarks: only well-matched feature points.
- No dense surface, no appearance model — and with one camera, no metric scale.



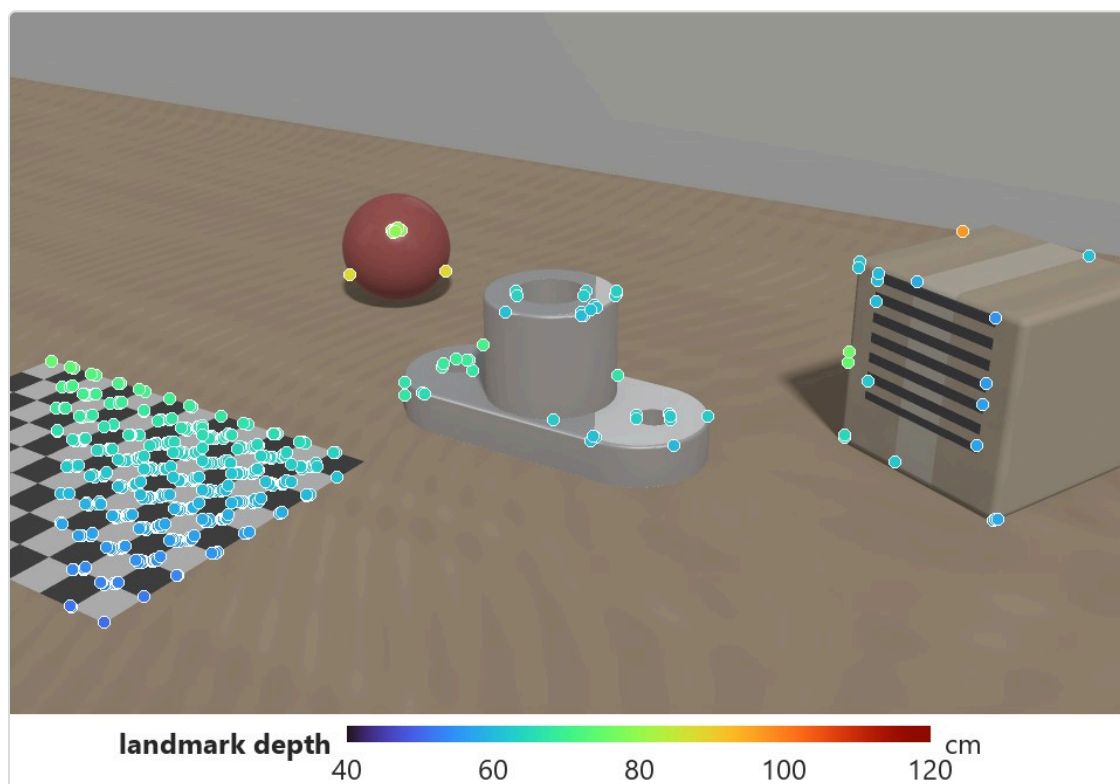
what VSLAM gives us

- ✓ camera poses (trajectory)
- ✓ sparse 3D landmarks
- ✓ keyframes, loop closures
- ✗ dense surface
- ✗ appearance, new views
- ✗ metric scale (monocular)

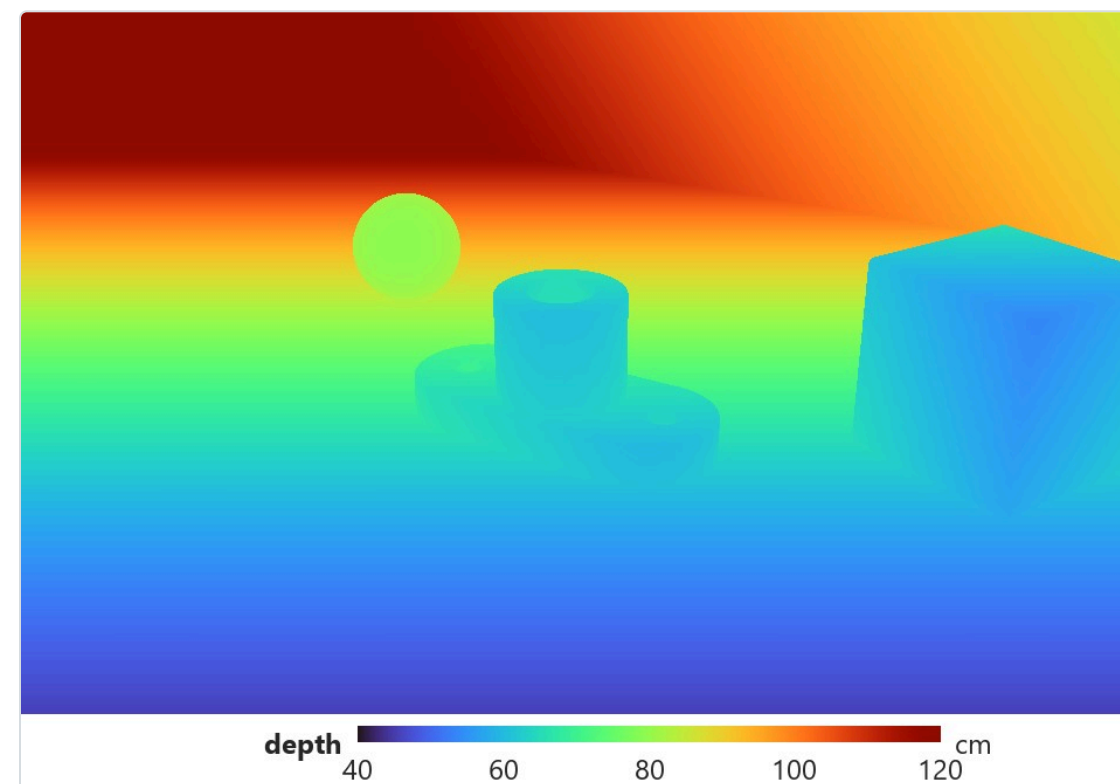
Sparse Map versus Dense Depth

Part 1 · From VSLAM to dense 3D

- Left: what a feature-based SLAM keeps — a few hundred well-matched points.
- Right: what we want — a depth value for every pixel, including smooth and shiny parts.
- Between them lies everything this lecture is about.



sparse: ORB feature points, coloured by depth



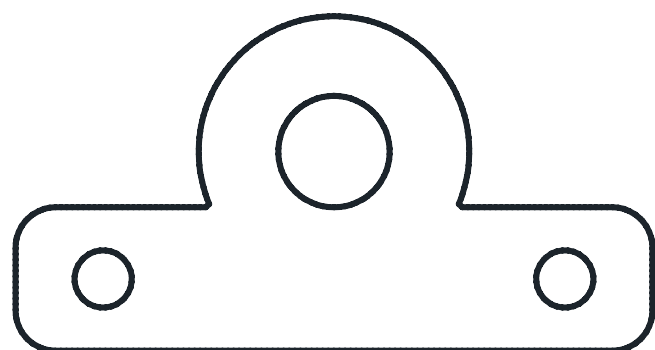
dense: ground-truth depth of the same view

What Reconstruction Should Deliver

Part 1 · From VSLAM to dense 3D

- Geometry: where surfaces and occupied space are.
- Appearance: how the scene looks from a given viewpoint, including reflections.
- Dynamics: how geometry and appearance change over time.

GEOMETRY



where is the surface?

APPEARANCE



how does it look from here?

DYNAMICS

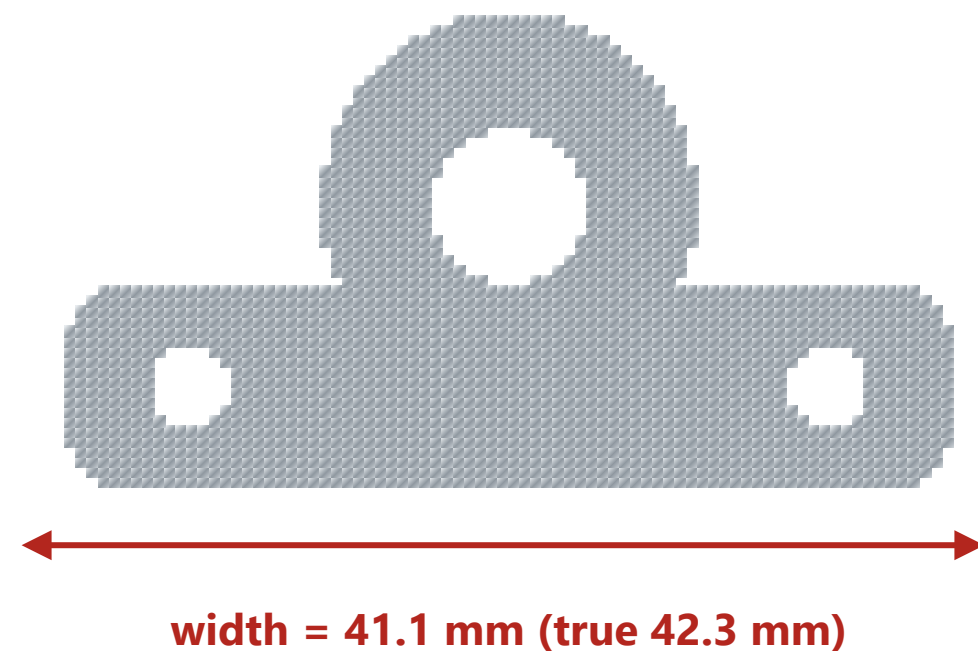
how does it change in time?

Machine Vision versus Computer Graphics

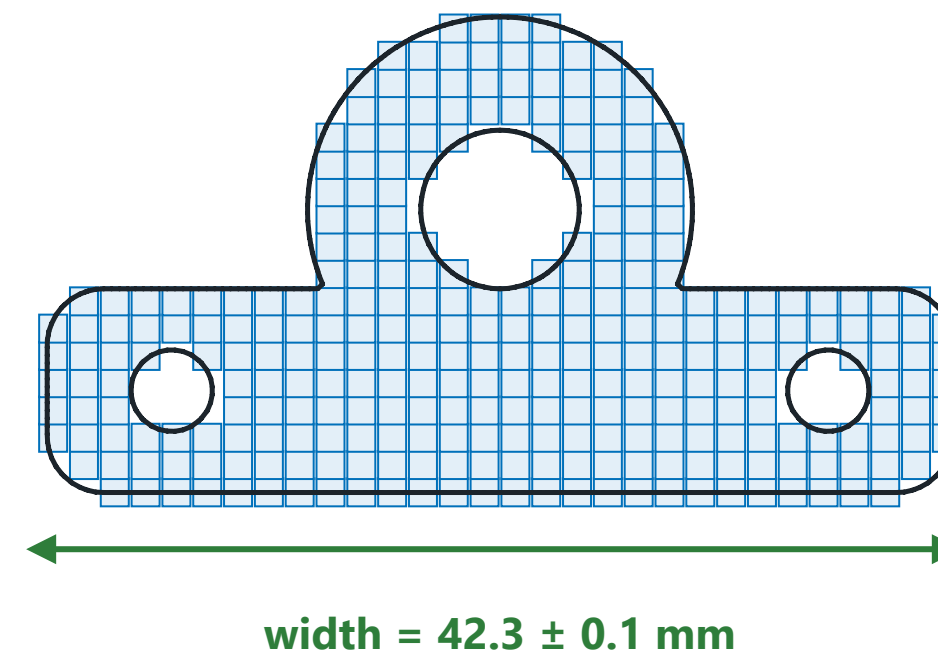
Part 1 · From VSLAM to dense 3D

- Graphics wants images that look right.
- Machine vision wants geometry that is right — in millimetres, with an uncertainty.
- A photorealistic model can still have the wrong dimensions.

photorealistic rendering



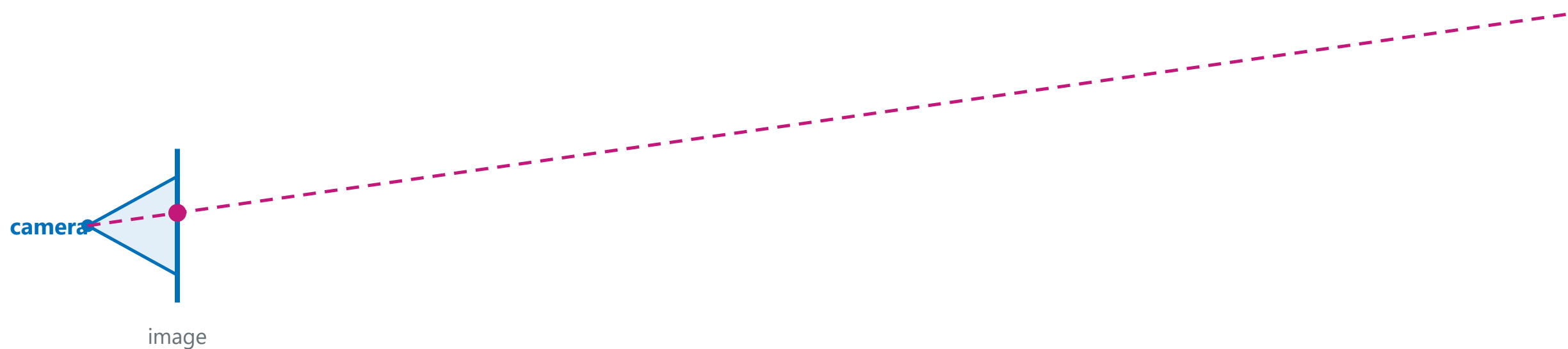
coarse, but measured



Recap: One Pixel Is a Ray

Part 1 · From VSLAM to dense 3D

- Projection $s \mathbf{u} = \mathbf{K} [\mathbf{R} \mid \mathbf{t}] \mathbf{X}$ maps a 3D point to a pixel.
- Inverting it gives a ray, not a point: depth is lost.
- Depth must come from more views, a sensor, or a learned prior.

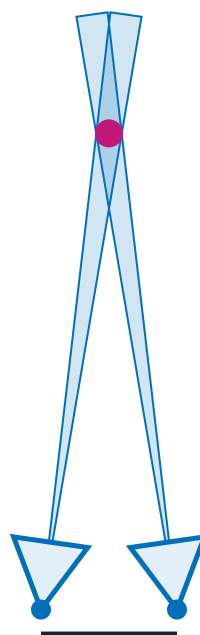


every point on the ray projects to the same pixel: one image does not determine depth

Recap: Triangulation Needs Baseline

Part 1 · From VSLAM to dense 3D

- Two rays from calibrated cameras intersect at the point.
- Each ray is uncertain by a fraction of a pixel; the overlap of the two wedges is the uncertainty.
- Small baseline or large depth: the overlap stretches along the viewing direction.



depth 7 m, baseline 2 m $\rightarrow$ depth uncertainty ≈ 21.4 cm

each ray is known only within
 $\pm 0.25^\circ$ (pixel noise);
the overlap is where the point can be

Your monocular VSLAM map looks great. Can you measure the width of a part with it?

Not yet: the map is sparse (few points on the part) and its scale is arbitrary without a metric reference (known size, stereo, IMU, depth sensor).



Six Ways to Represent the Same Object

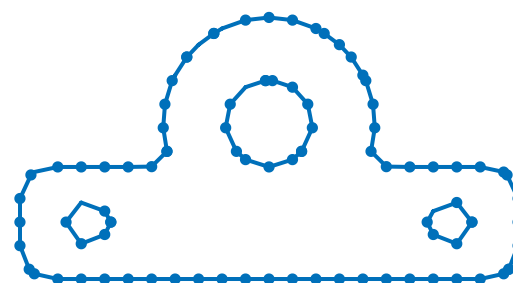
Part 2 · Representing 3D

- We use one running example: a cross-section of a flange (2D for clarity; the ideas are 3D).
- Explicit: points, mesh, voxels, Gaussians. Implicit: fields queried at any point.
- The choice decides what can be measured, rendered, and stored.

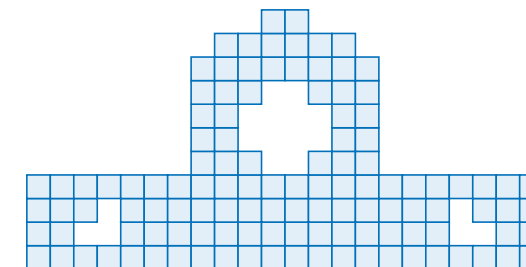
point cloud



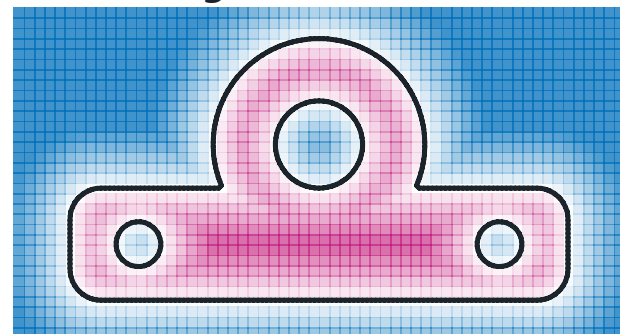
mesh



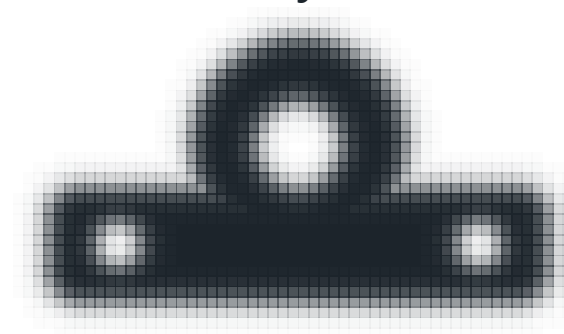
voxels



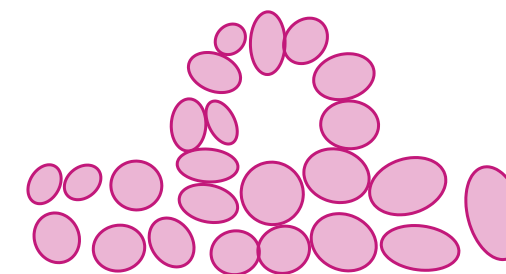
signed distance



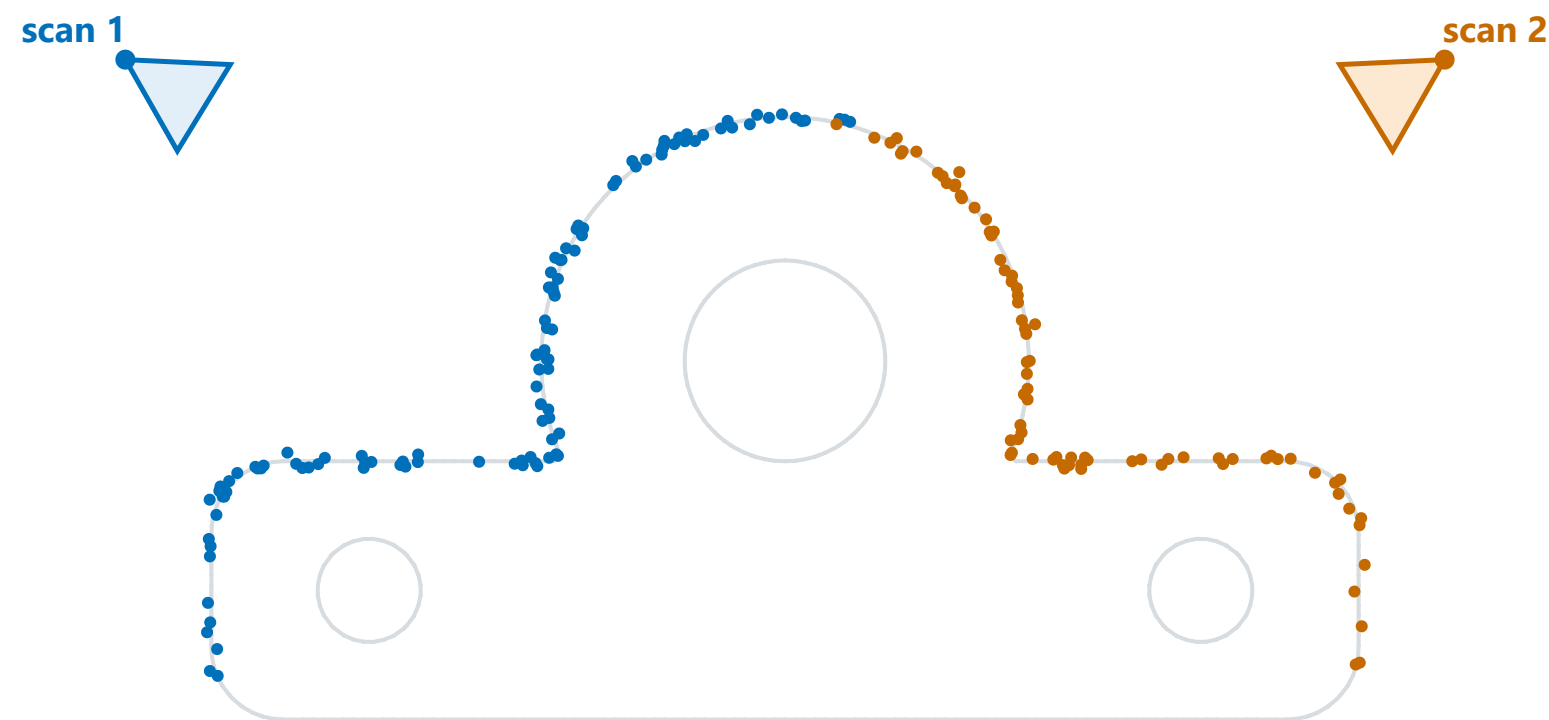
density field



Gaussians



- A set of 3D points, optionally with colour and normal: what LiDAR and triangulation produce.
- Simple and flexible, but no explicit surface and no inside/outside.
- Only what a sensor saw is represented; gaps stay gaps.

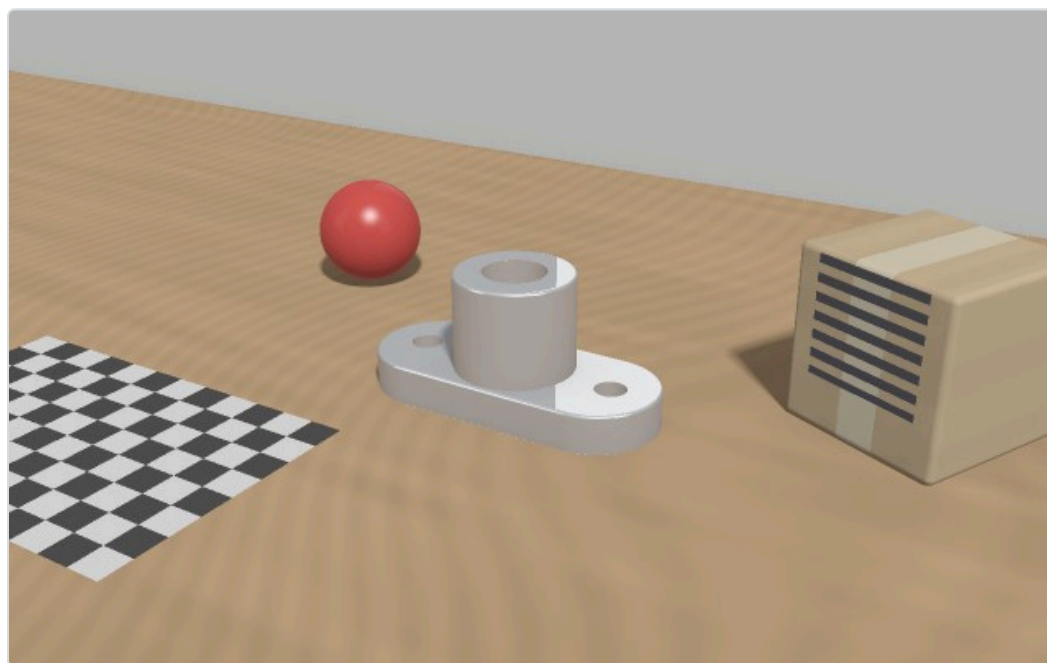


only surfaces seen by a sensor get points; the bottom is never observed

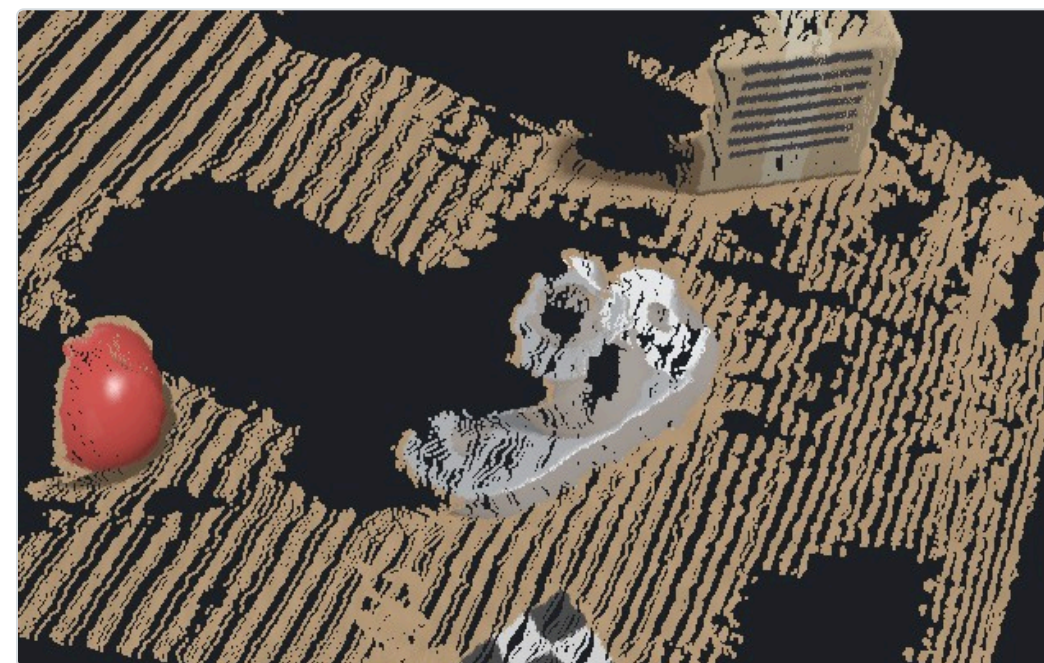
A Real Point Cloud, Seen from Another Angle

Part 2 · Representing 3D

- Depth from one stereo camera, turned into points and viewed from above.
- Behind every object there is a shadow of missing data; the plain wall is gone entirely.
- A single view is never enough — and points alone do not tell what is solid.

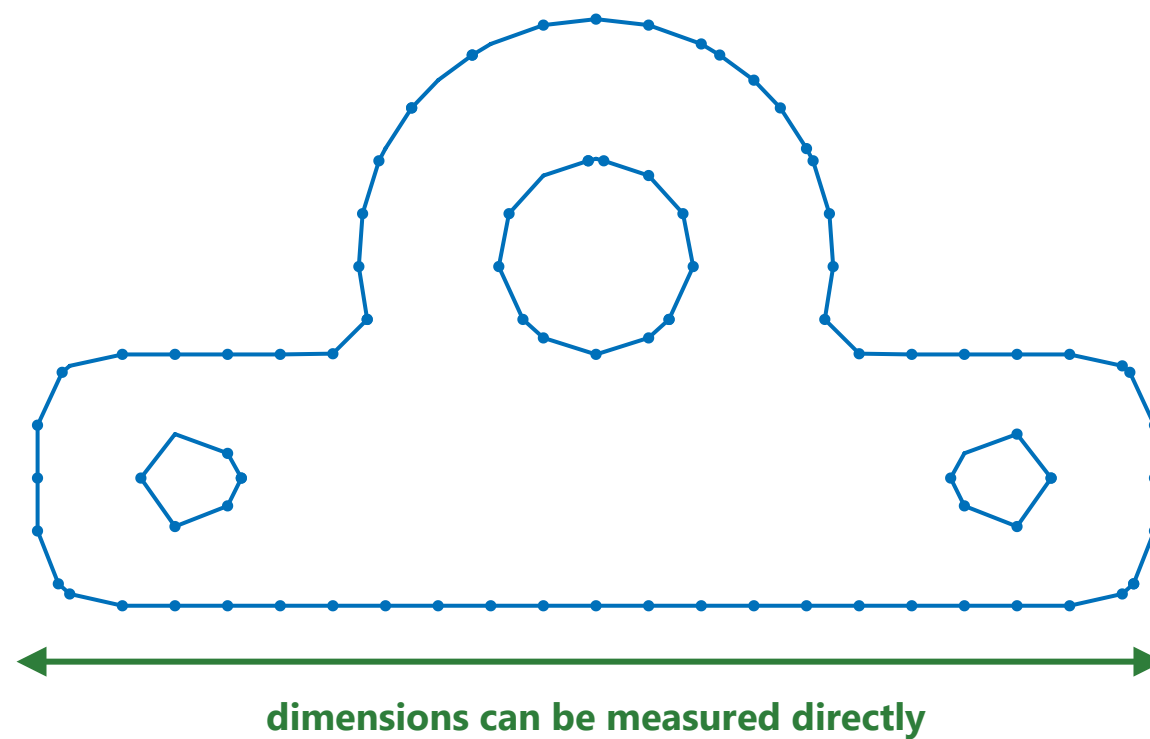


what the camera saw



its point cloud from another viewpoint

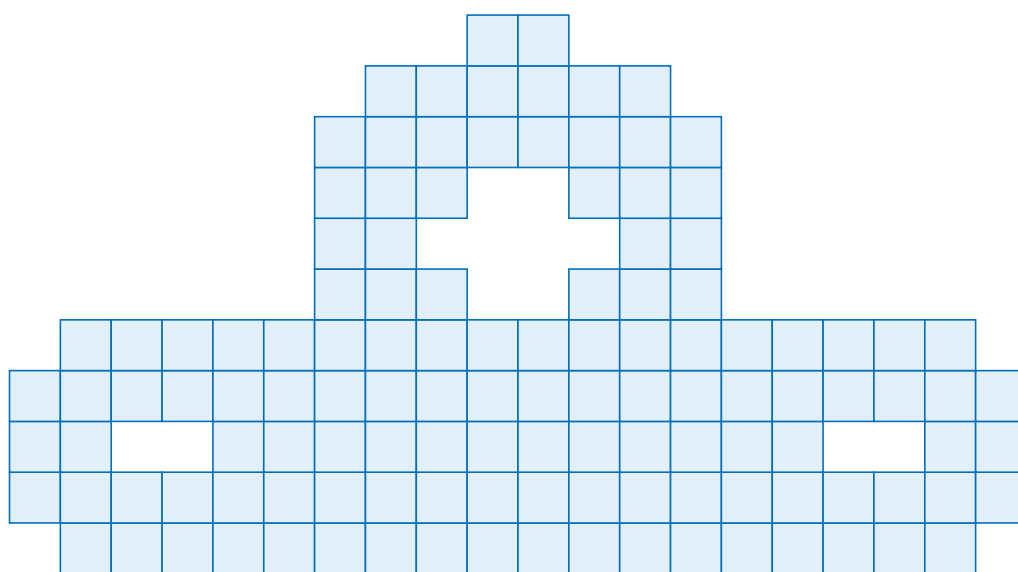
- Vertices connected into triangles describe the surface explicitly.
- The standard for measurement, collision checking, and classical rendering.
- Hard to build directly from images; changing topology is difficult.



2D analogy: vertices + edges
(3D: vertices + triangles)

explicit surface: inside/outside,
normals, collisions, rendering

- A voxel is the 3D analogue of a pixel: a cell of a regular grid.
- It can store occupancy, distance, colour, or a feature vector.
- Memory grows with the cube of the resolution — try the slider.

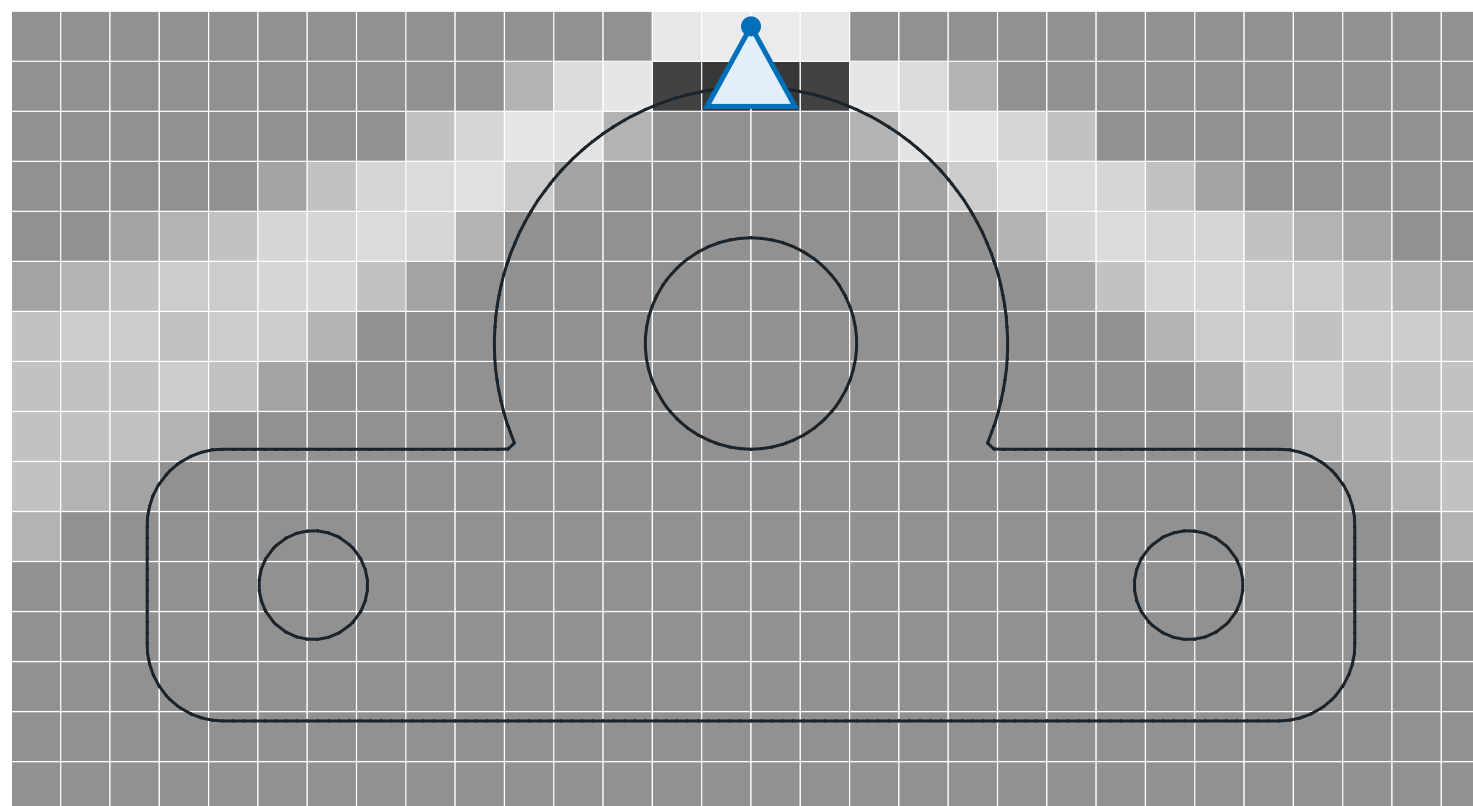


the same resolution in 3D

24³	0.014 M cells	0.1 MB
128 ³	2.1 M cells	8.4 MB
256 ³	16.8 M cells	67 MB
512 ³	134.2 M cells	537 MB

float32 per cell; 2× resolution = 8× memory

- Each cell holds the probability that it is occupied.
- Every measured ray updates two kinds of cells: those it passes (free) and the one it hits.
- The standard map for robot navigation and planning.



for every measured ray:

cells it passes: more likely free
(log-odds - 0.4)

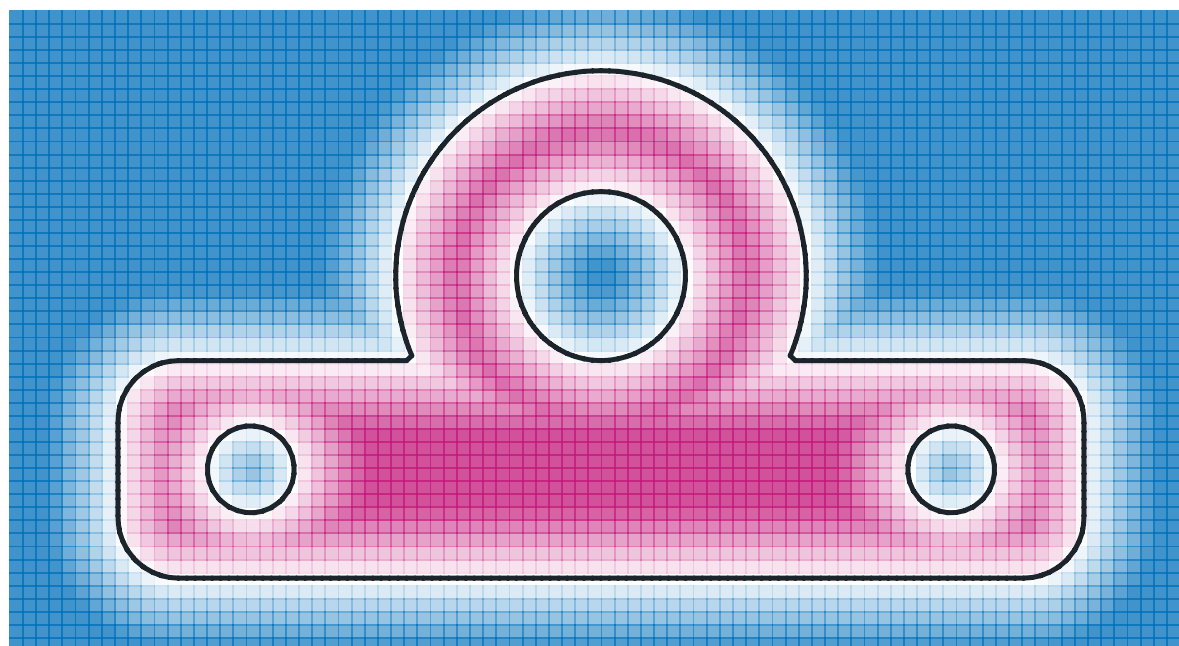
cell where it hits: more likely
occupied (log-odds + 0.85)

free space is information too

Signed Distance and TSDF

Part 2 · Representing 3D

- Each cell stores the distance to the nearest surface, negative inside.
- The surface is where the distance is zero — sub-voxel accurate by interpolation.
- TSDF keeps only a band around the surface and averages many depth maps (KinectFusion).



signed distance to the surface

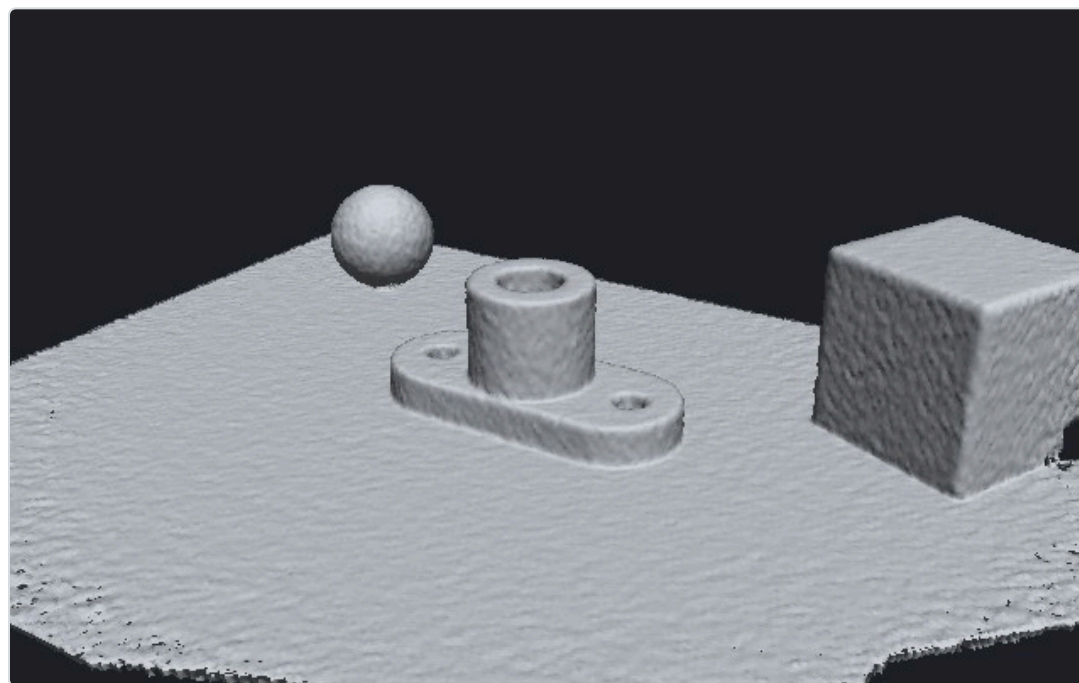
- outside (+)
- inside (−)

the surface is the zero level set;
far from it the value is truncated
to $\pm \mu$: only a thin band is stored

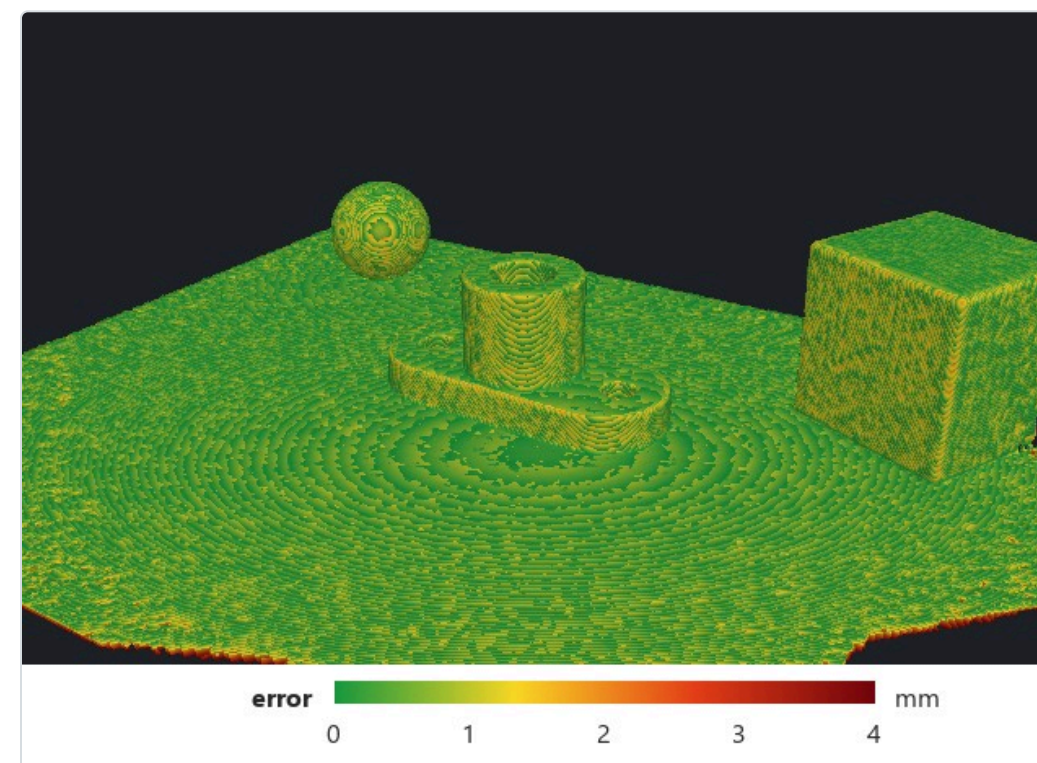
Fusing Many Noisy Depth Maps

Part 2 · Representing 3D

- 14 depth maps from different viewpoints, each with realistic sensor noise, fused into one TSDF.
- Averaging over views removes most of the noise: the surface is smooth and closed where it was seen.
- Right: the same surface coloured by its distance to the true object — the kind of map used in metrology.

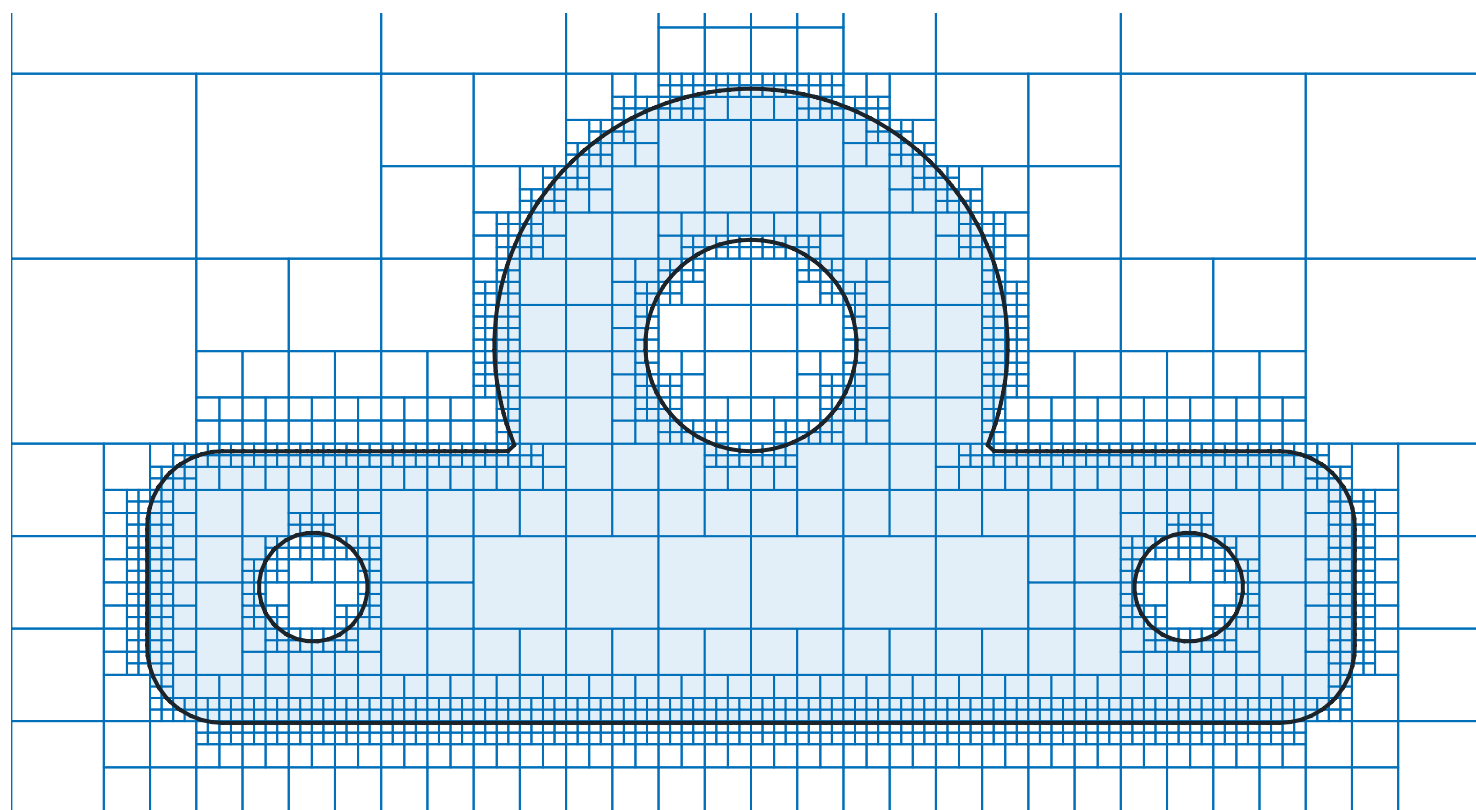


fused surface (shaded)



accuracy map: green = within a fraction of a millimetre

- Most of space is empty; storing it all wastes memory.
- Quadtrees / octrees refine only where the surface is.
- Hash grids map a huge virtual grid into a compact table (used again for fast NeRFs).



depth 7: 1804 cells vs 16384 in a full grid

split a cell only where the
surface passes through it
(3D: octree)

- A neural network $f(\mathbf{x})$ returns occupancy or signed distance for any 3D point.
- Continuous: no fixed resolution; the scene is stored in the network weights.
- The price: every query is a network evaluation.

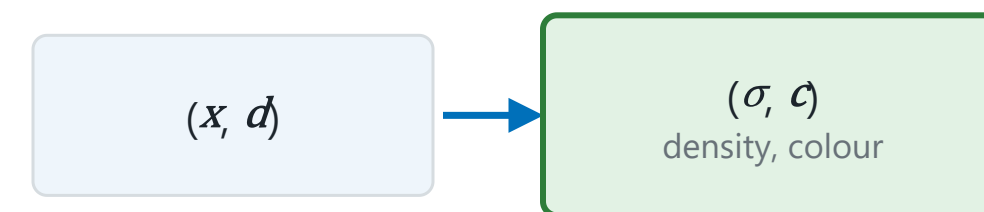


query any point, at any resolution;
the surface is where $f = 0$
the whole scene is stored in θ

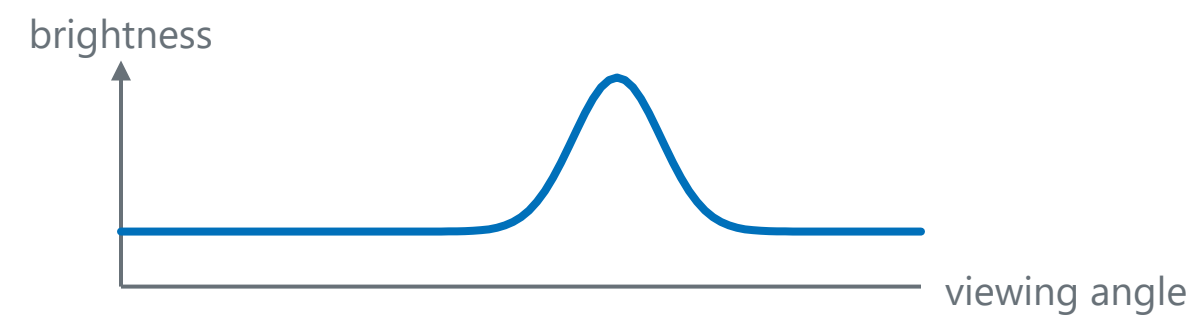
Radiance Fields: Colour Depends on Direction

Part 2 · Representing 3D

- A radiance field returns density and colour for a position *and* a viewing direction.
- It can model reflections and gloss that a single texture cannot.
- It describes appearance — not automatically a surface.



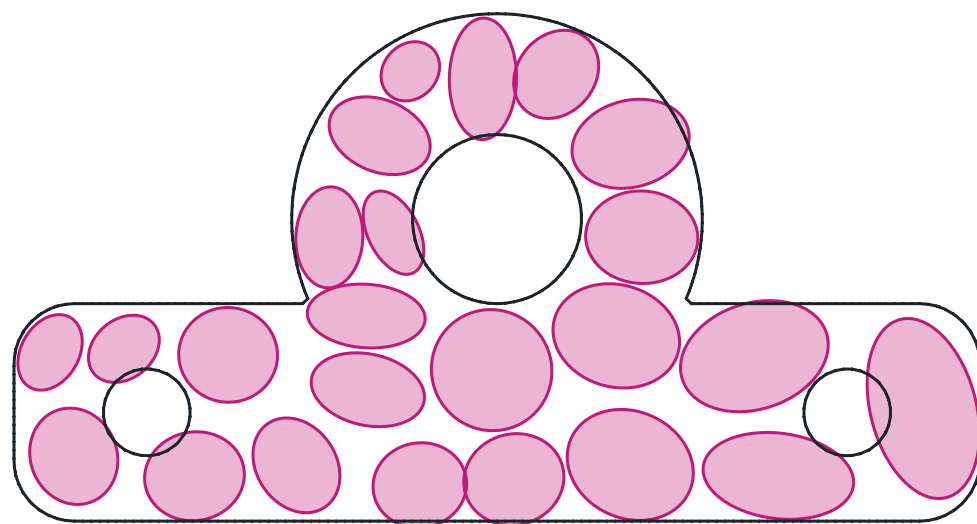
the same 3D point has a different
colour from different directions
(glossy, metallic surfaces)



Gaussians as Primitives

Part 2 · Representing 3D

- Cover the scene with many soft, oriented ellipsoids.
- Each has a centre, a covariance, an opacity, and a colour.
- Explicit like a point cloud, but continuous and fast to render.



each primitive: centre, covariance
(size + orientation), opacity, colour

anisotropic ellipses follow thin
and elongated structures; more
primitives = finer detail

Which Representation for Which Job?

Part 2 · Representing 3D

representation	measure dimensions	plan / collide	render new views	memory
point cloud	roughly	with effort	poorly	moderate
mesh	yes	yes	yes (with textures)	small
occupancy / voxels	at grid resolution	yes	poorly	large (cubic)
SDF / TSDF	yes (zero level)	yes	via mesh	large, sparse helps
radiance field (NeRF)	indirectly	indirectly	yes	small (weights)
3D Gaussians	with care	indirectly	yes, real time	large

Choose by the downstream task. Conversions are possible (field → mesh, Gaussians → surface), but each one adds error.

Check question

Choose a representation

You must (a) check a hole tolerance, (b) plan a robot path, (c) show a customer the line from new viewpoints. Which representation for each?

(a) a mesh or SDF validated against ground truth; (b) an occupancy grid; (c) a radiance field or Gaussian splats.



The Classical Dense Pipeline

Part 3 · Learned depth

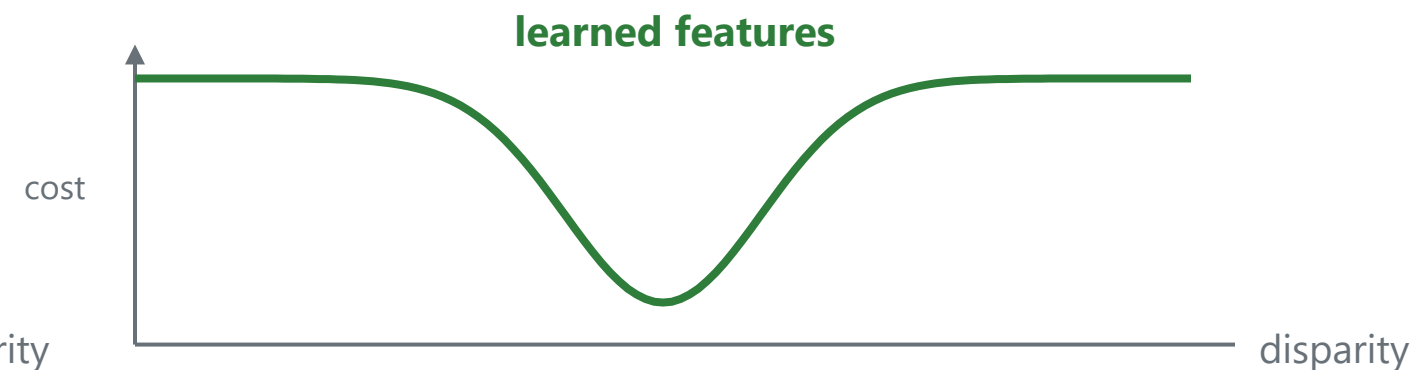
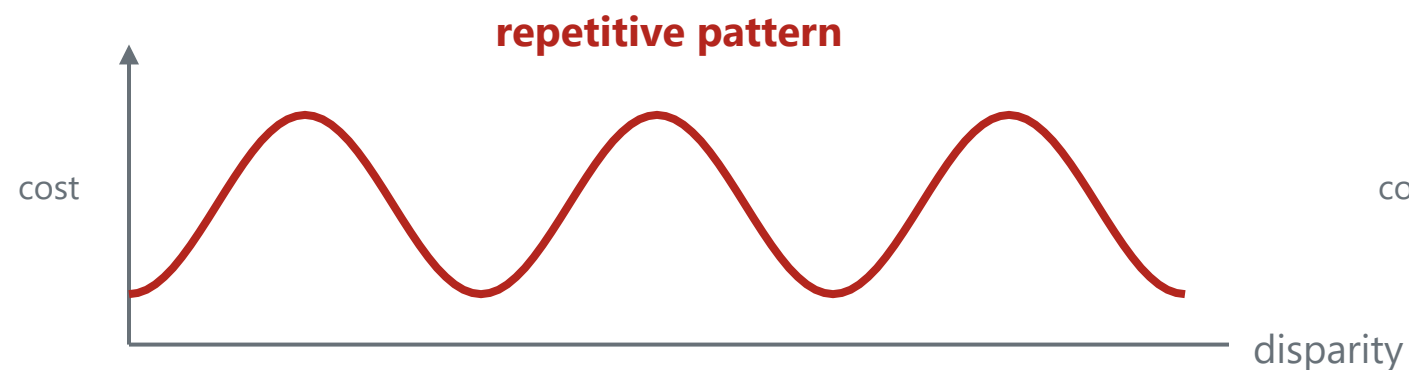
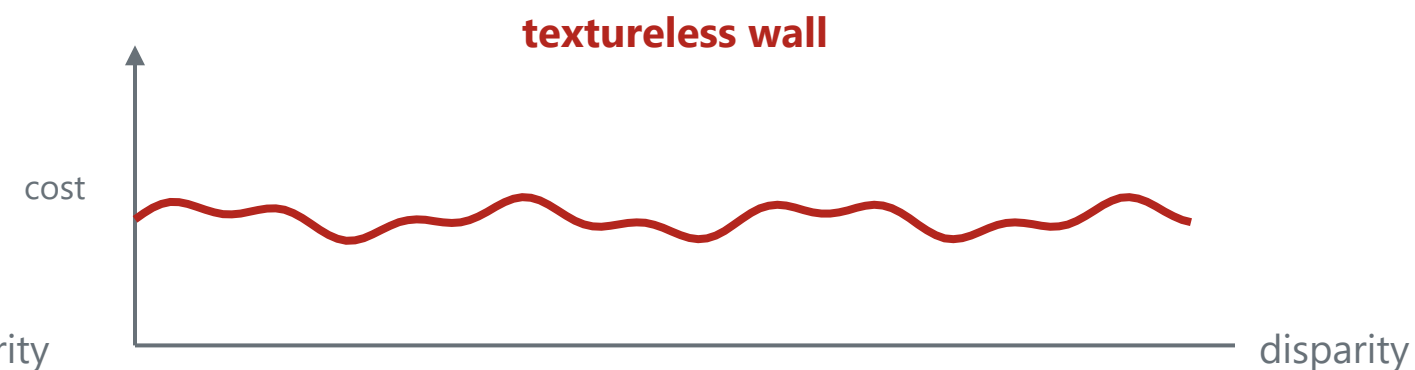
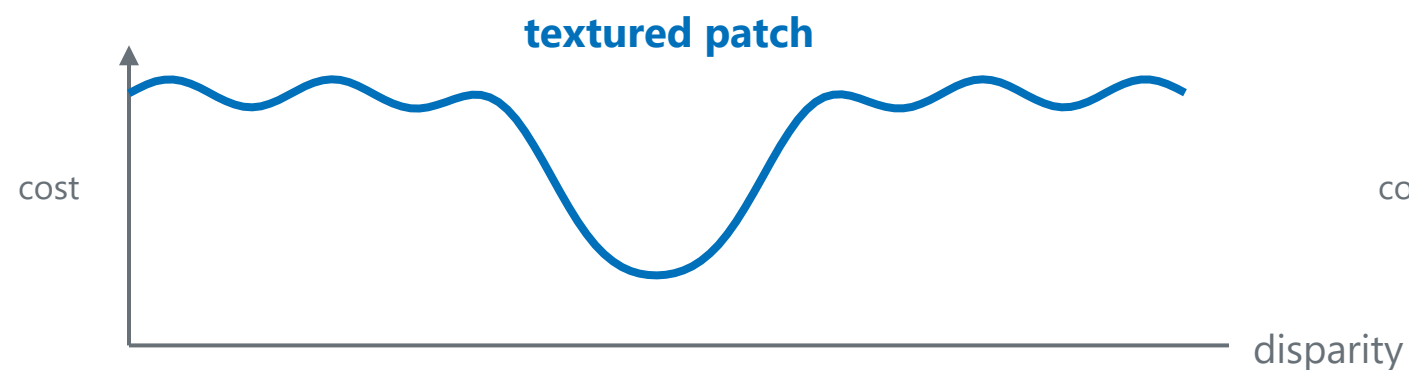
- Poses from SLAM/SfM, then a depth map for each view, then fusion into one model.
- The depth-map step is where most errors appear.
- It is also where learned matching helps most.



Where Classical Matching Fails

Part 3 · Learned depth

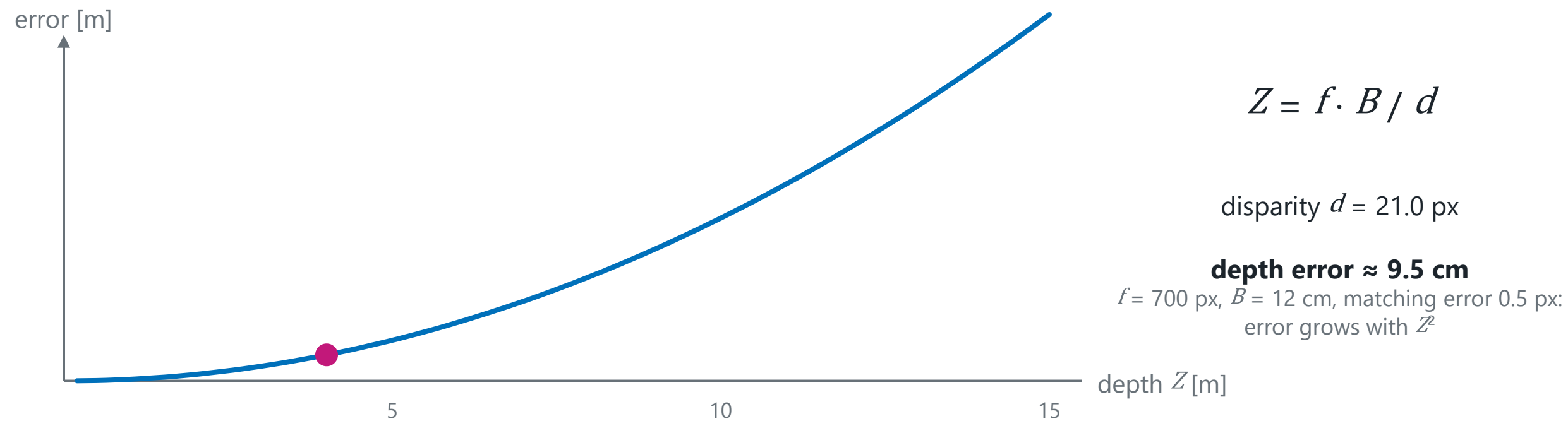
- Matching compares small patches along the epipolar line and picks the lowest cost.
- Textureless or repetitive regions give flat or periodic cost curves: no clear answer.
- Learned features use context (lecture 2) and produce one clear minimum.



Stereo: Depth from Disparity

Part 3 · Learned depth

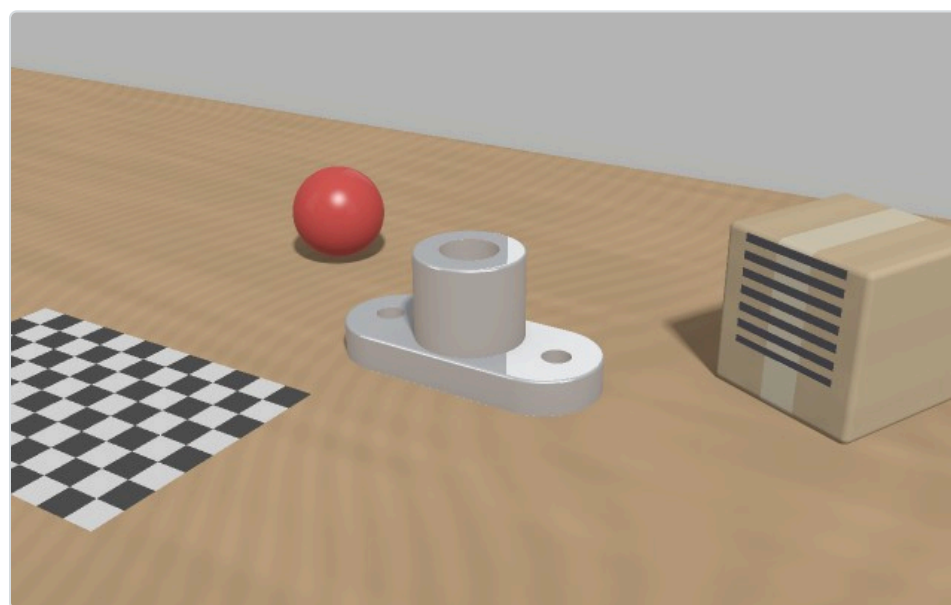
- For a rectified pair, depth is inversely proportional to disparity: $Z = fB / d$.
- A fixed matching error in pixels becomes a depth error growing with Z^2 .
- Better matching (learning) helps, but baseline and focal length set the limits.



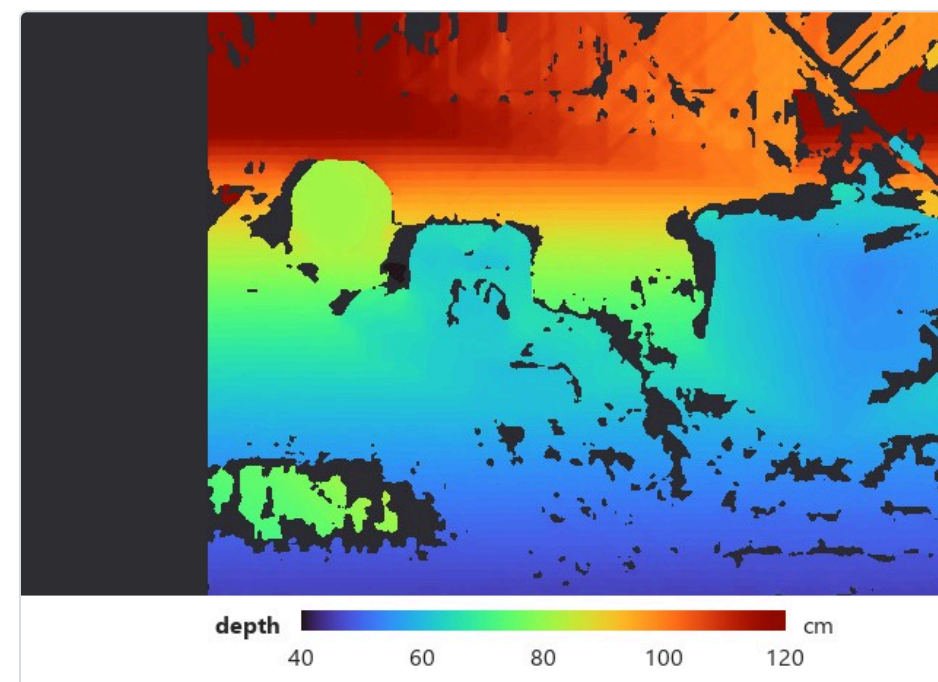
Classical Stereo on Our Scene

Part 3 · Learned depth

- Two cameras 6 cm apart; a classical matcher (OpenCV SGBM) finds the disparity for every pixel.
- Black = no reliable match. The plain wall and the shiny metal are mostly missing.
- Where it does match, the depth is good: median error 3 mm.



left image (right camera 6 cm aside)

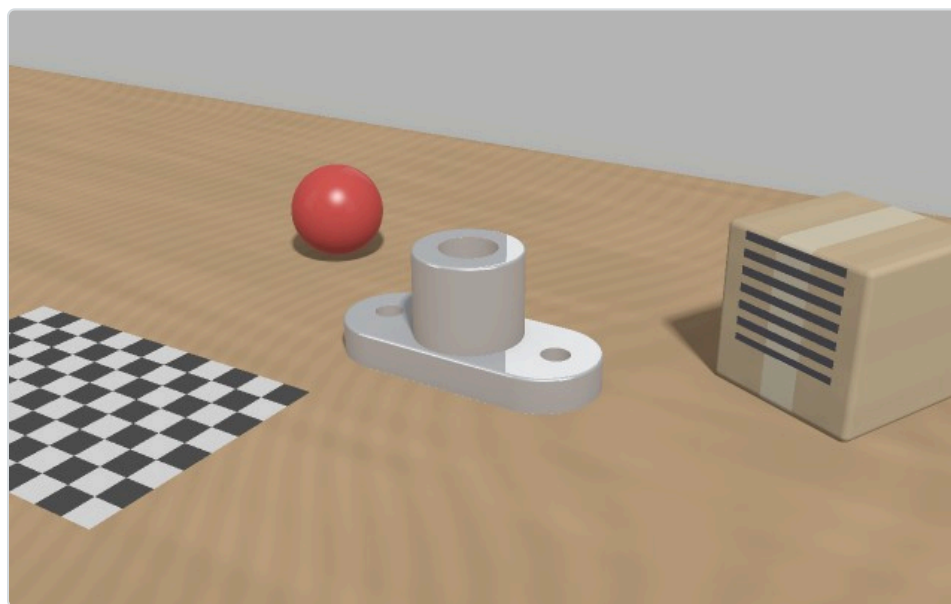


depth from stereo matching

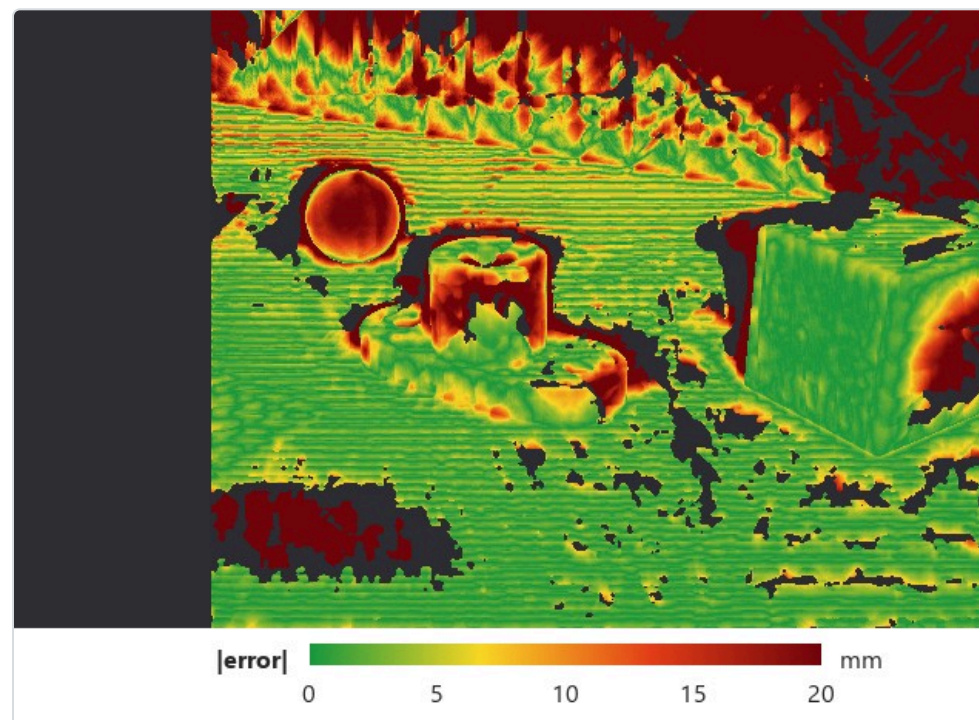
Where Stereo Is Wrong

Part 3 · Learned depth

- The same result coloured by its error against the true depth, in millimetres.
- Red regions are exactly the troublemakers: textureless wall, repetitive checkerboard, object edges.
- This is why learned matching exists: it uses context to fill and fix these areas.

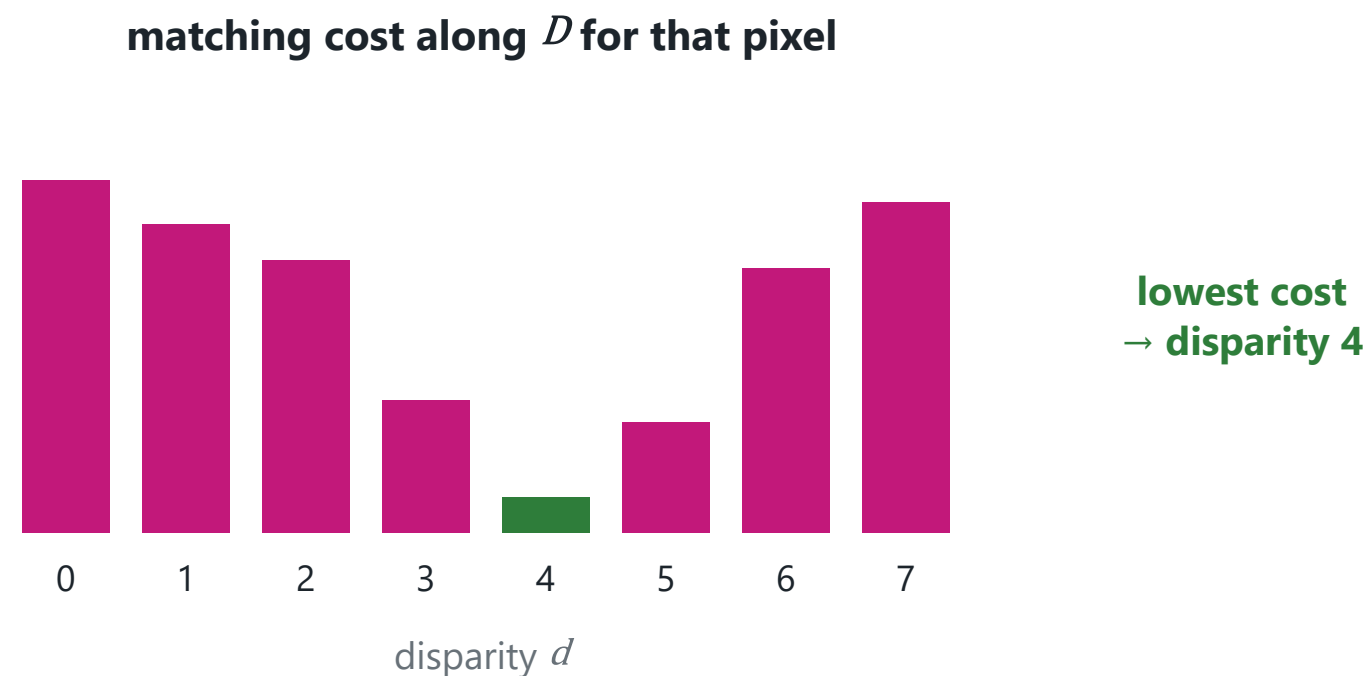
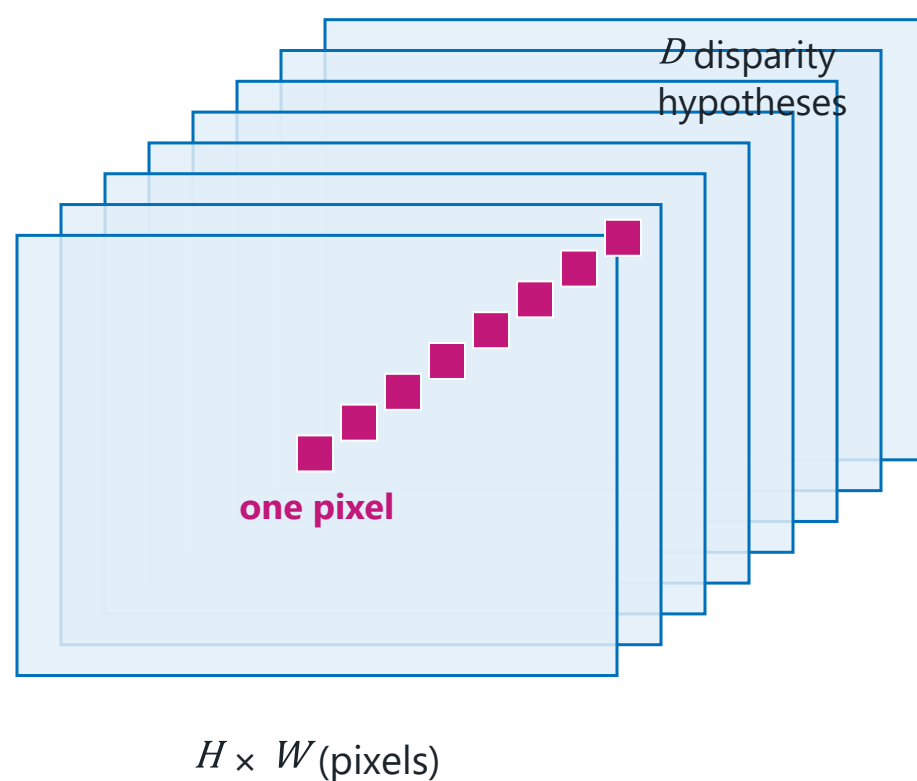


input



error map: green ≤ 5 mm, dark red ≥ 20 mm

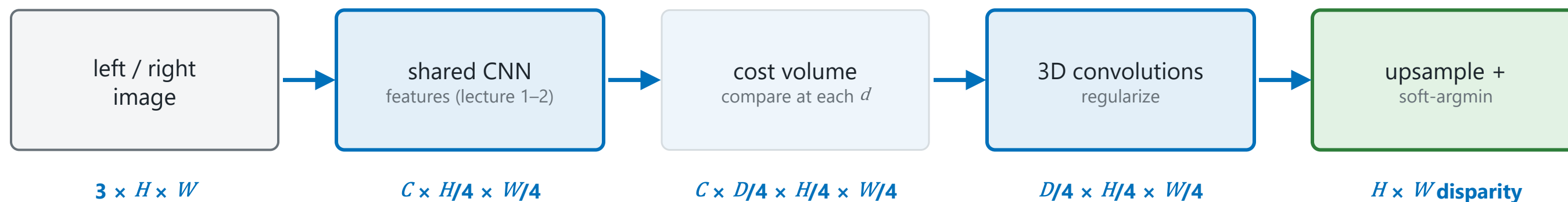
- For every pixel and every candidate disparity, store how well the two images match.
- The result is a 3D tensor: height $\times$ width $\times$ disparity.
- The best disparity of a pixel is the minimum along its column.



A Learned Stereo Network

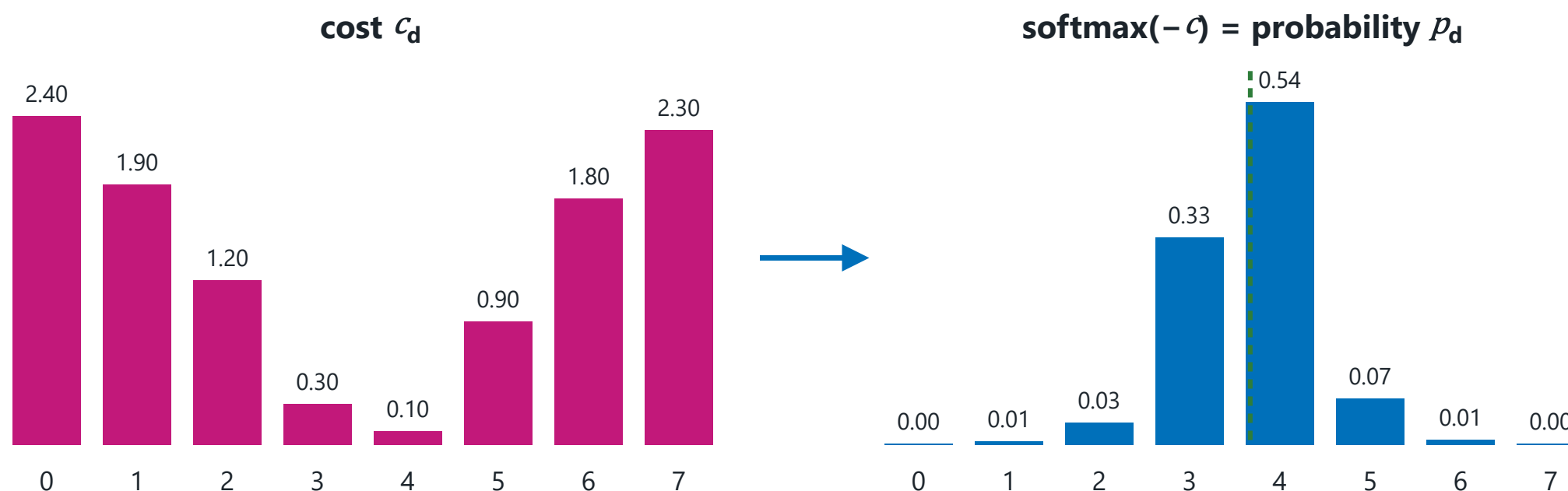
Part 3 · Learned depth

- A shared CNN extracts features from both images.
- Features are compared at each disparity to build a cost volume; 3D convolutions clean it up.
- A differentiable readout turns it into a disparity map.



every step is differentiable: the whole network is trained end to end on disparity

- A hard argmin has no useful gradient and is limited to whole disparities.
- Softmax over negative costs gives a probability per disparity.
- The expected disparity is sub-pixel and can be trained end to end.



$$\hat{d} = \sum d \cdot p_d$$

$$= 3.67$$

sub-pixel and differentiable,
unlike a hard argmin

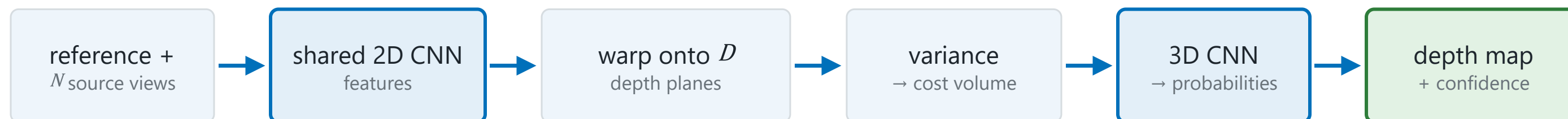
Multi-View Stereo: The Plane Sweep

Part 3 · Learned depth

- Hypothesize planes at many depths in front of the reference camera.
- Project the other views onto each plane and measure how much they disagree.
- At the true depth all views agree — the minimum of the curve.



- MVSNet-style networks do the plane sweep on CNN features instead of pixels.
- The variance across views forms a cost volume; a 3D CNN regularizes it.
- Output: a depth map with a confidence map, for every reference view.

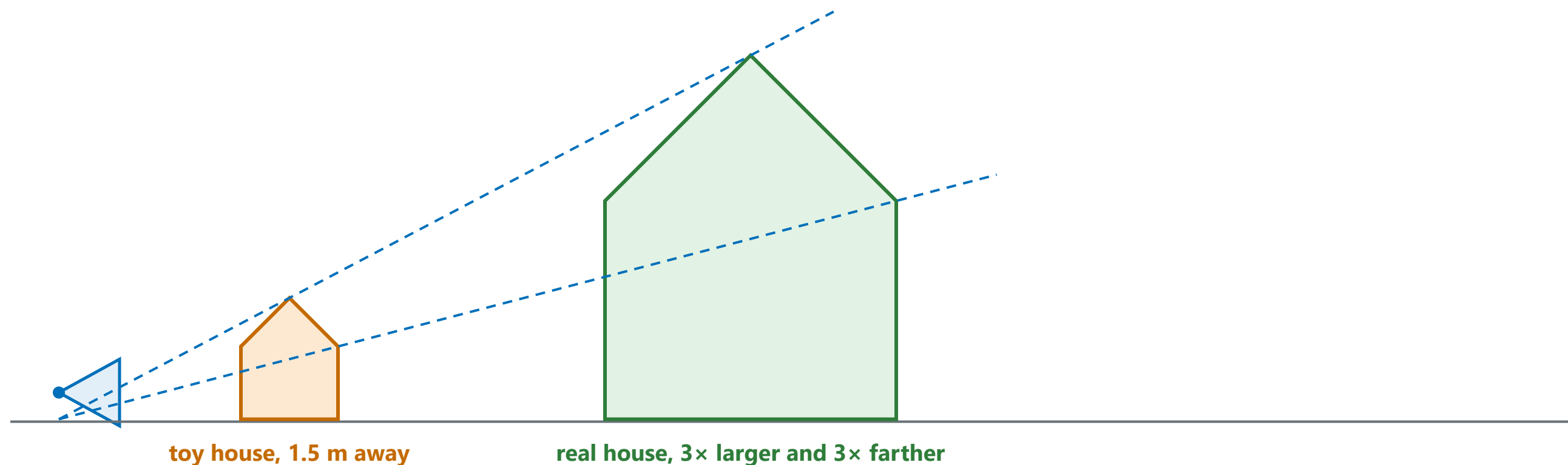


depth maps of many reference views are then filtered by confidence and fused (as in the classical pipeline)

Monocular Depth and the Scale Problem

Part 3 · Learned depth

- A network can predict depth from one image using learned priors about the world.
- Geometry alone cannot tell a toy from a house: the scale is ambiguous.
- Good relative depth is not yet a metric reconstruction.



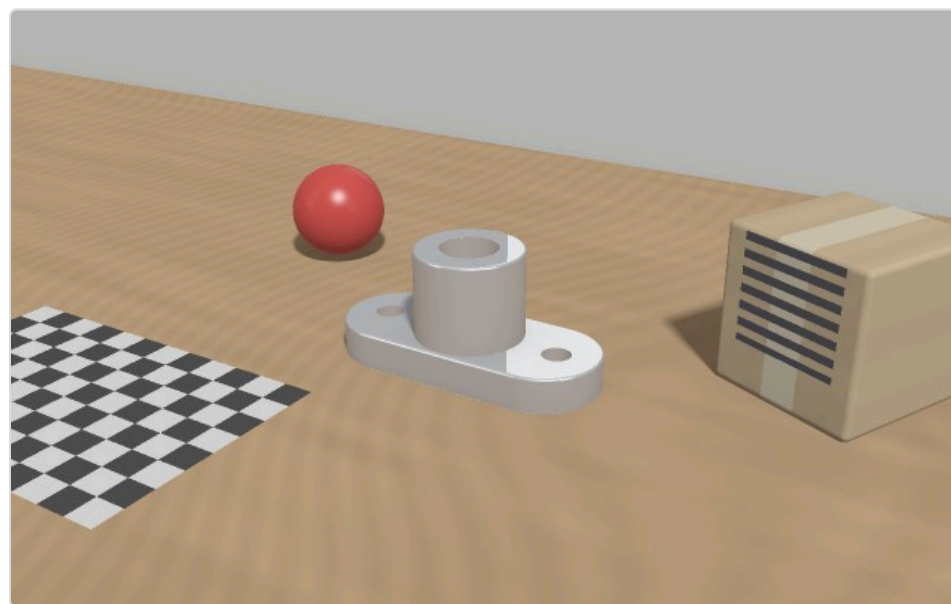
same image, different scenes: a single image fixes depth only up to scale

monocular networks learn plausible relative depth; metric scale needs a reference (SLAM points, a known size, IMU)

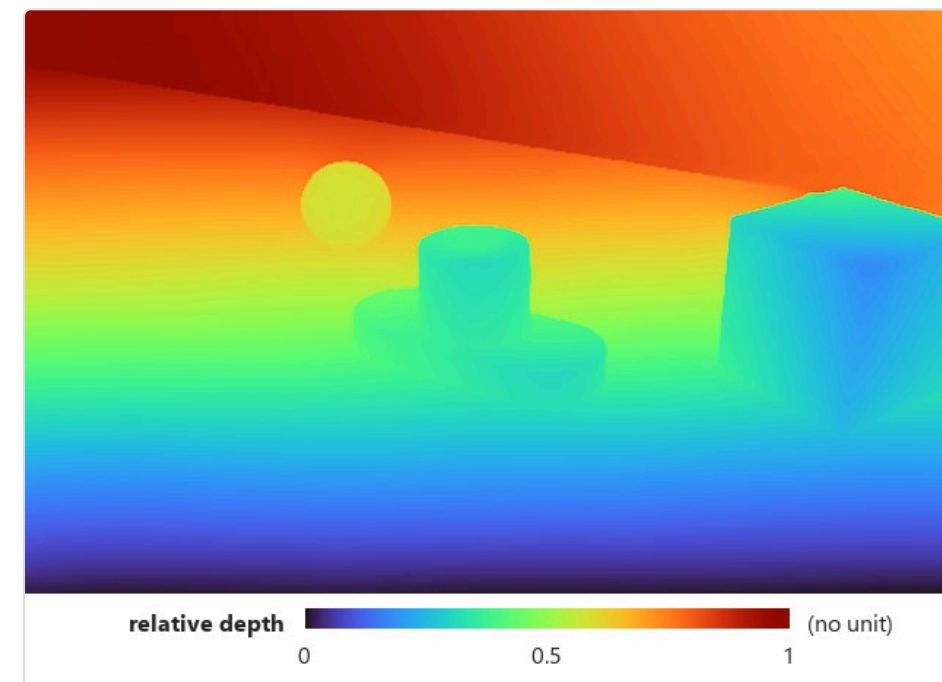
A Depth Network Sees One Photo

Part 3 · Learned depth

- A pretrained network (Depth Anything V2) predicts depth from the left image alone.
- No holes, sharp object outlines, even the plain wall — learned from millions of photos.
- But the output has no unit: it is only relative depth.



one input image

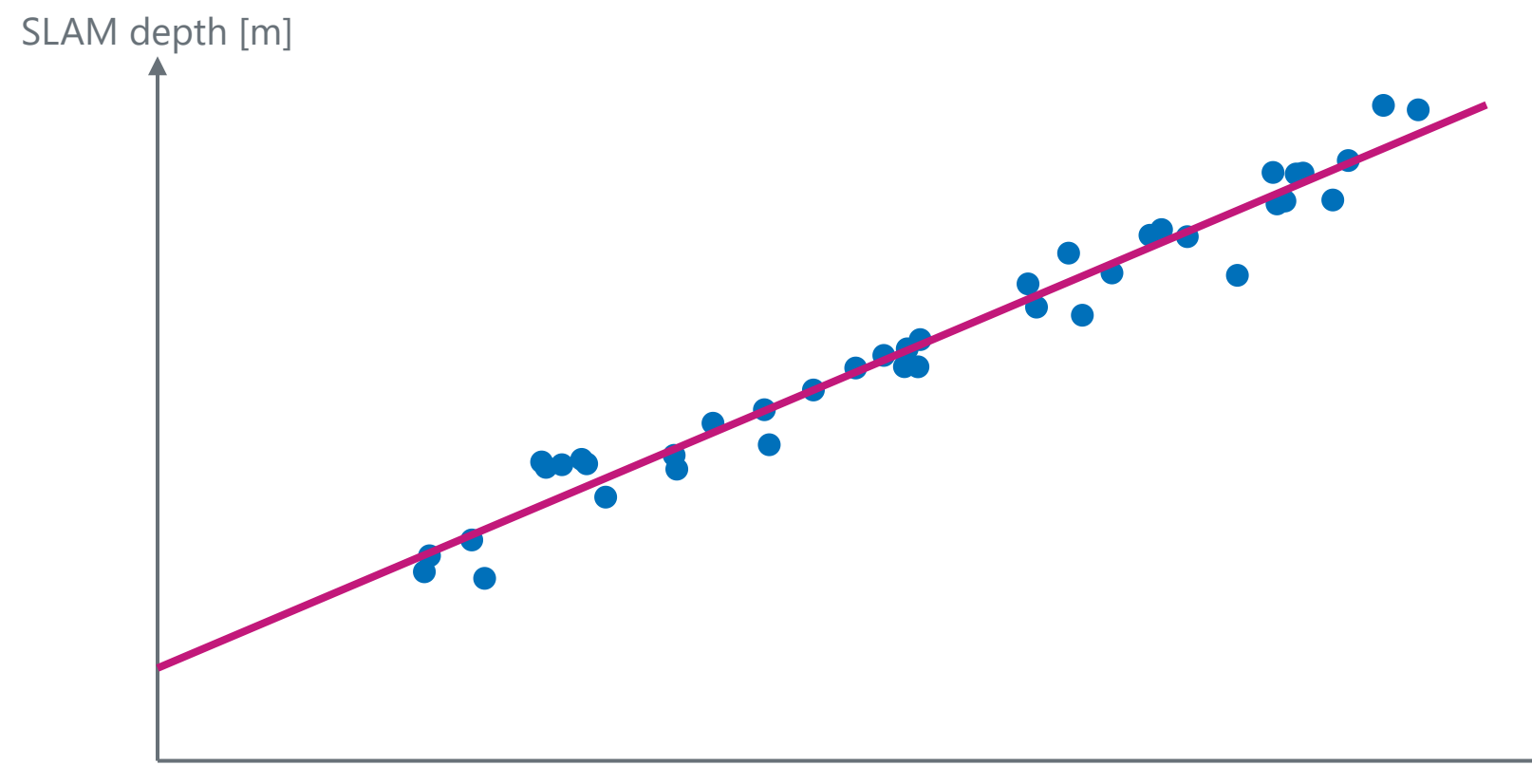


network output: dense, but without scale

Fixing Scale with SLAM Points

Part 3 · Learned depth

- VSLAM landmarks give a few reliable depths in each keyframe.
- Fit a scale and shift between network depth and landmark depth.
- Dense learned depth then inherits the scale of the SLAM map.



least squares on sparse
SLAM points: $s = 2.58$, $t = 0.42$

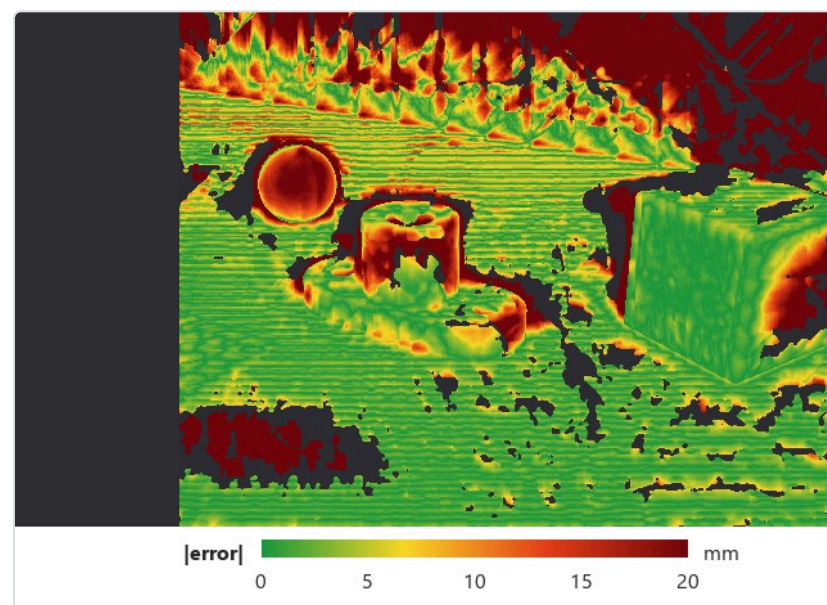
metric depth = $s \cdot$ network depth + t
for every pixel of the image

each dot: a pixel where VSLAM
has a triangulated landmark

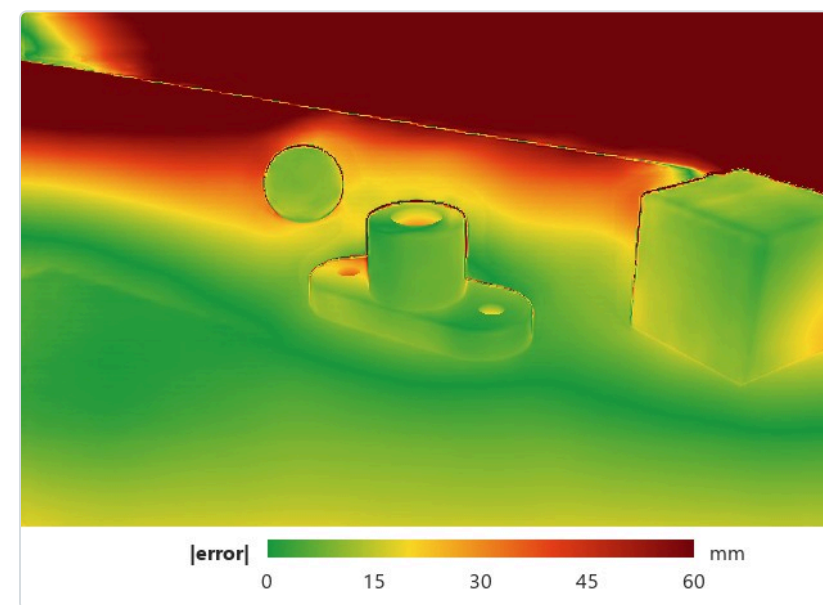
After Scaling: Complete, but Less Accurate

Part 3 · Learned depth

- Scaled with the sparse SLAM points, every pixel gets a depth in centimetres.
- Compare: stereo has holes but ≈ 3 mm error; the network is complete but ≈ 12 mm off.
- Typical practice: learned depth for completeness, geometry for accuracy — often combined.



stereo: holes, accurate where valid (0–20 mm)



network + scale: no holes, larger errors (0–60 mm)

**Your stereo depth is accurate at 1 m but poor at 8 m.
Why, and what would help?**

The error grows with $Z^2/(fB)$: $64 \times$ worse at 8 m. Increase the baseline or focal length, use more views (MVS), or move the camera closer.

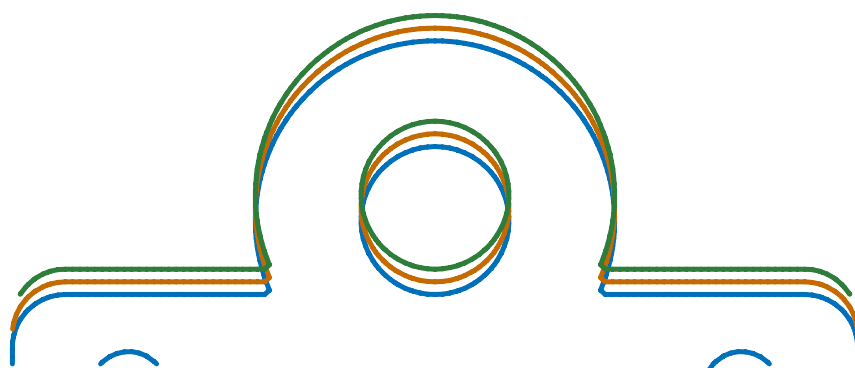


From Depth Maps to One Scene Function

Part 4 · Neural radiance fields

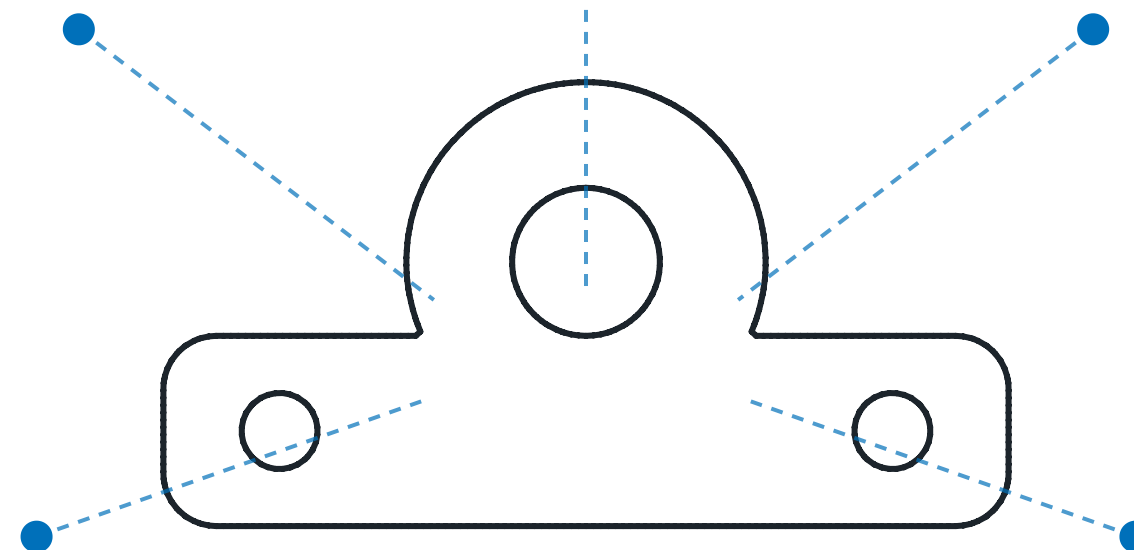
- MVS estimates a depth map per view and fuses them afterwards.
- A neural field instead optimizes one function for the whole scene.
- A rendering loss ties every pixel of every camera to that single model.

MVS: one depth map per reference view



separate estimates must be fused and may disagree

neural field: one function for the whole scene

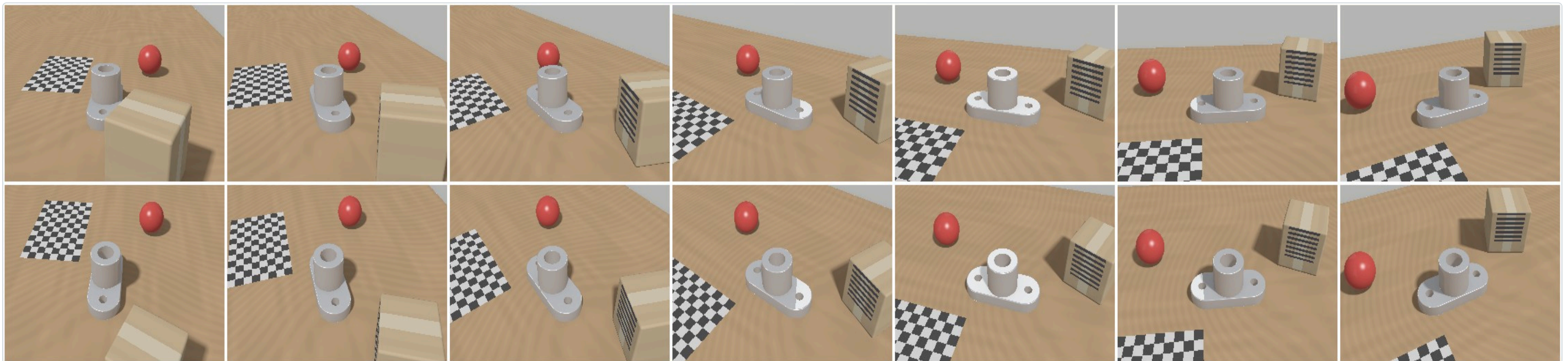


every pixel of every camera constrains the same model

What a NeRF Needs: Many Photos with Known Poses

Part 4 · Neural radiance fields

- Typically 50–200 photos around the object; here 14 for illustration.
- The pose of every photo must be known — from VSLAM or structure from motion.
- Output: a model that renders the scene from any viewpoint between these cameras.

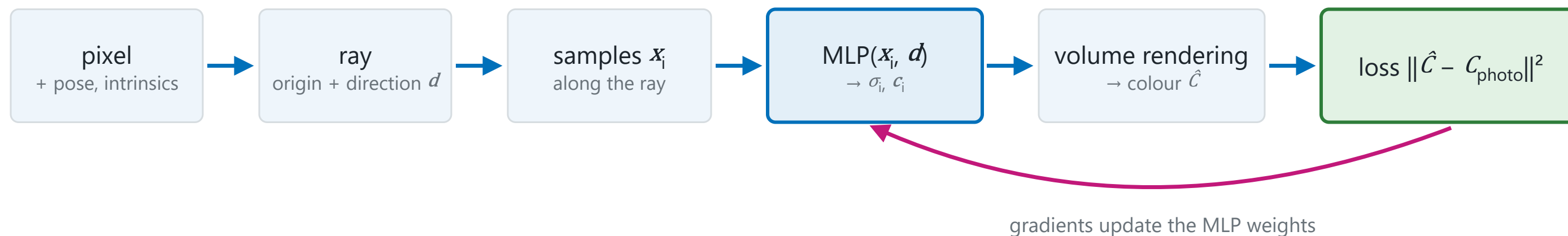


training images from two rings of camera positions

NeRF: What Goes In and What Comes Out

Part 4 · Neural radiance fields

- For a pixel, the pose and intrinsics give a ray; we sample points along it.
- An MLP maps each sample's position and the ray direction to density and colour.
- Volume rendering turns the samples into one pixel colour, compared with the photo.



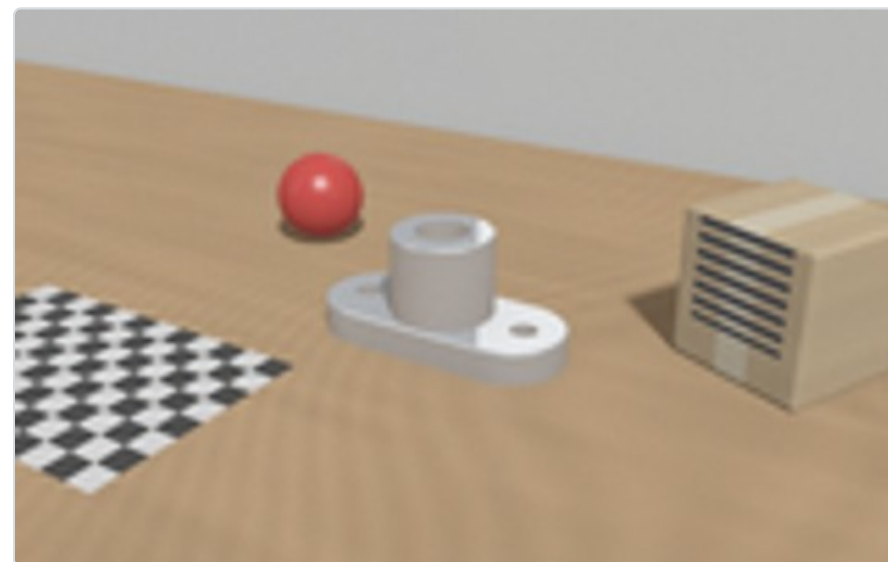
Why NeRF Needs a Trick for Detail

Part 4 · Neural radiance fields

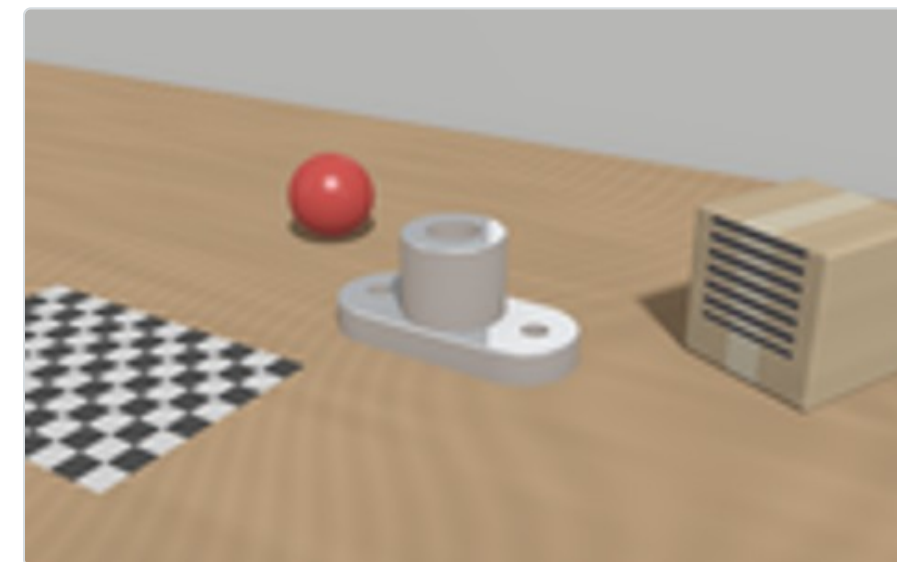
- We trained the same small network to reproduce the photo from pixel coordinates.
- Fed raw (x, y) , it only learns a blurry image. Fed sines and cosines of the coordinates, it learns every edge.
- NeRF uses the same trick in 3D — without it, the scene would be blurry.



raw coordinates · PSNR 23 dB

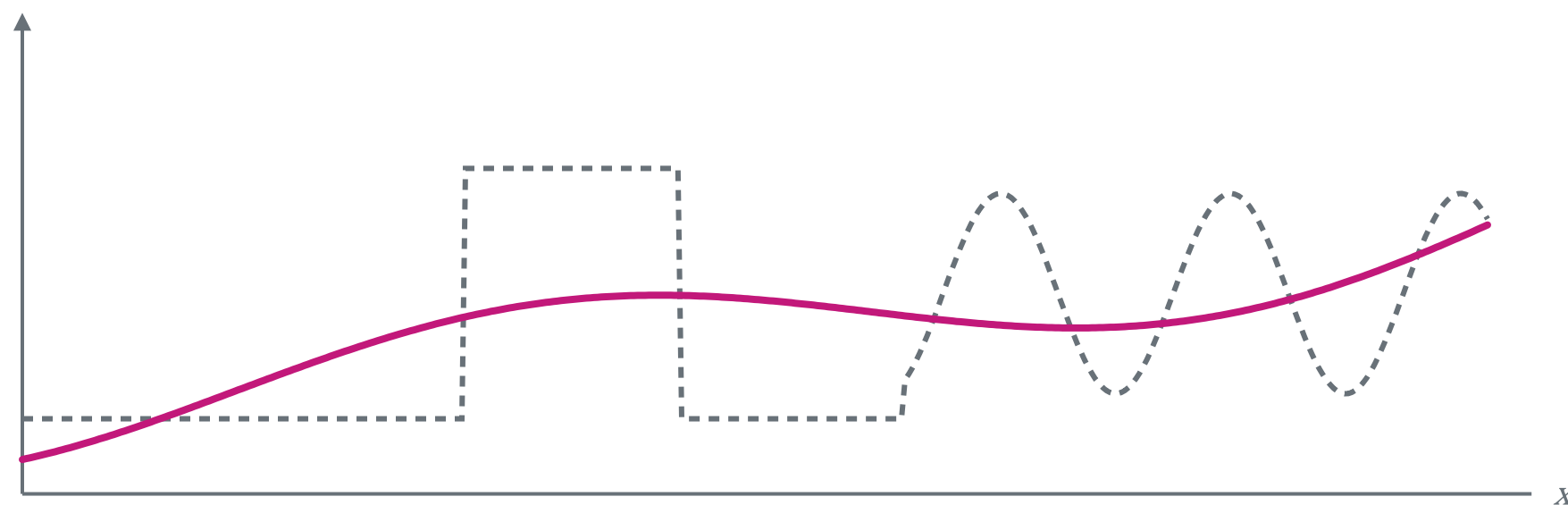


with positional encoding · PSNR 43 dB



target photo

- MLPs fed raw coordinates learn only smooth, blurry functions.
- Encoding each coordinate with sines and cosines of rising frequency exposes fine detail.
- Move the slider: the same fitting problem with more frequencies.



L = 2 frequencies

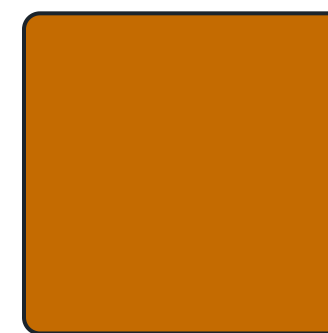
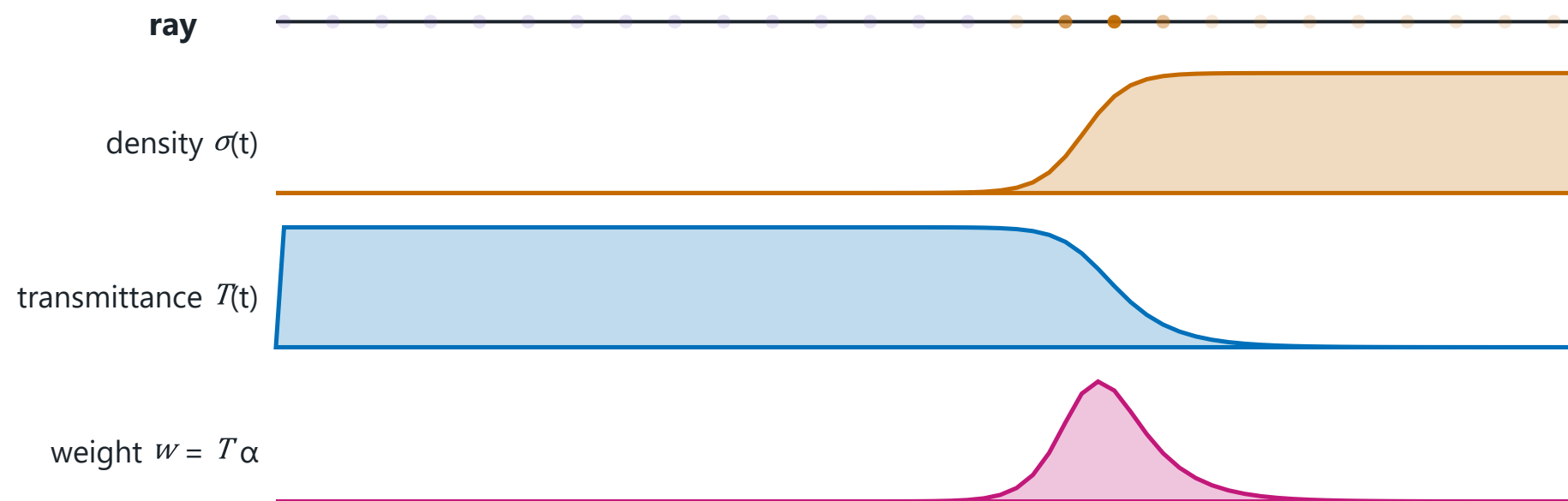
$\gamma(x) = [\sin(2^k \pi x), \cos(2^k \pi x)]_{k < L}$
fed to the network instead of x

curve: least-squares fit on $\gamma(x)$;
few frequencies = blurry, more = detail

Volume Rendering Along a Ray

Part 4 · Neural radiance fields

- Density says how likely the ray is to stop at each sample.
- Transmittance says how much light is still unblocked; it only decreases.
- Weights = transmittance $\times$ opacity peak at the first surface; they mix the colours.



rendered pixel

$\sum w = 1.00$, background 0.00

- Discrete form: $\hat{\mathcal{C}} = \sum T_i \alpha_i c_i$, with $\alpha_i = 1 - e^{-\sigma_i \delta}$ and $T_i = \prod_{j < i} (1 - \alpha_j)$.
- Every step is differentiable, so the photo loss trains density and colour.
- Four samples, computed column by column.

sample i	σ_i	$\alpha_i = 1 - e^{-\sigma_i \delta}$	$T_i = \prod (1 - \alpha_j)$	$w_i = T_i \alpha_i$	c_i	$w_i c_i$
1	0	0.000	1.000	0.000	0.9	0.000
2	0.5	0.118	1.000	0.118	0.6	0.071
3	8	0.865	0.882	0.763	0.2	0.153
4	2	0.393	0.119	0.047	0.4	0.019

$$\hat{\mathcal{C}} = \sum w_i c_i + T_{\text{end}} \cdot \text{background} = 0.242 + 0.072 \cdot 1.0 = \mathbf{0.314}$$

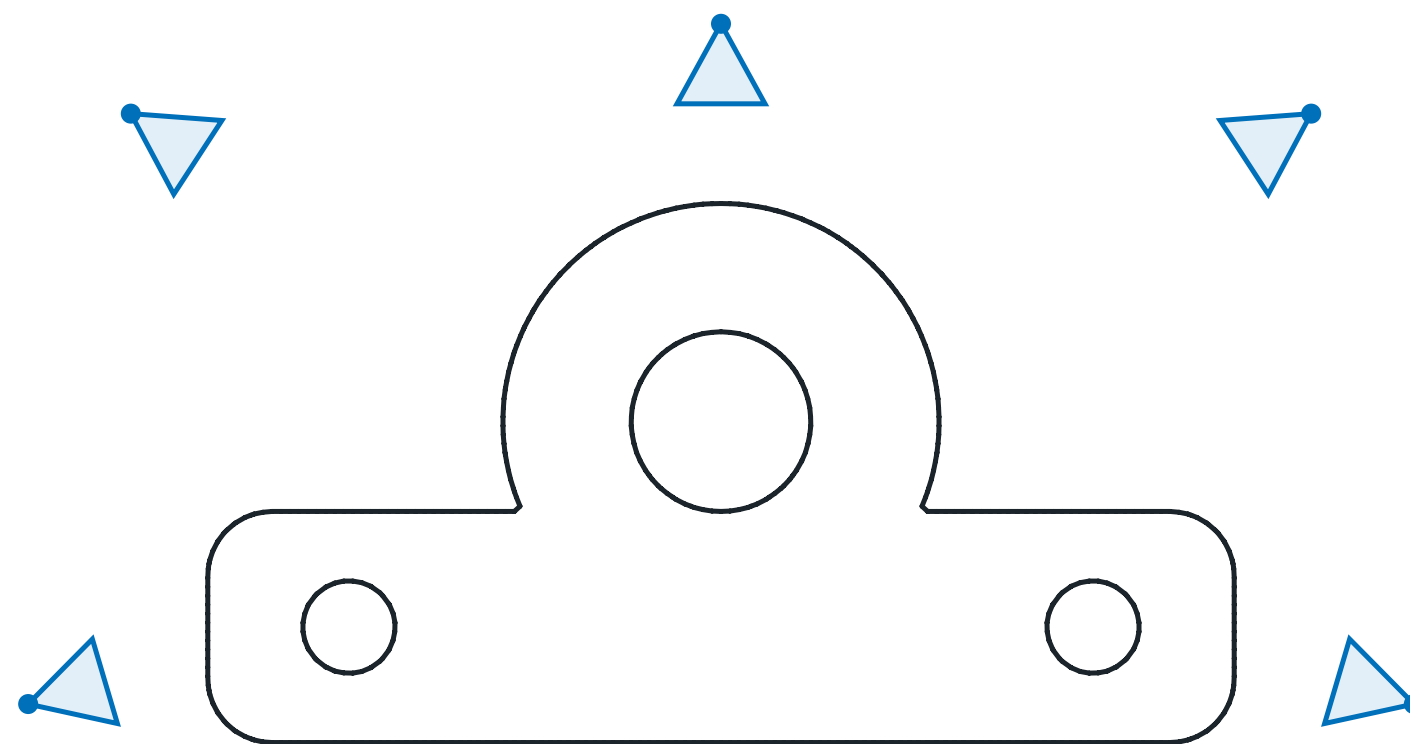
sample 3 (high density) dominates: it is the first opaque thing the ray meets

step $\delta = 0.25$

Training a NeRF

Part 4 · Neural radiance fields

- Each iteration draws a random batch of rays from all training images.
- Render them, compare with the observed pixels, back-propagate into the MLP.
- This is optimization of one scene, not learning that generalizes to other scenes.

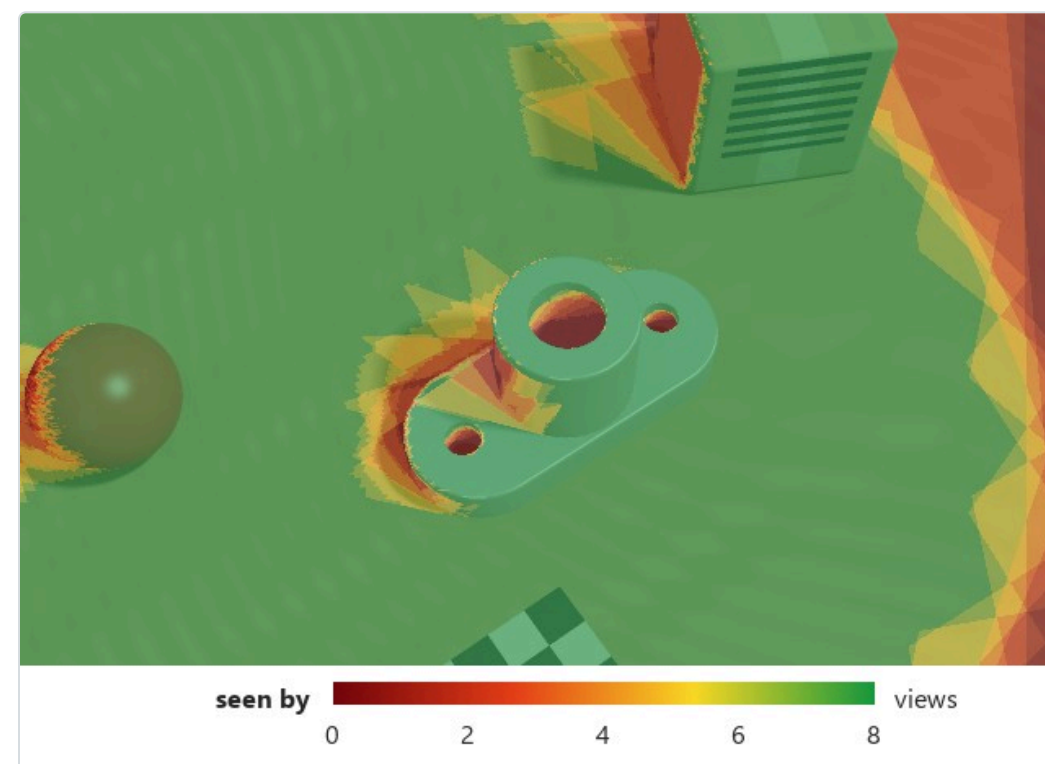


each step: a random batch of rays
from all training images

Which Parts Did the Cameras Actually See?

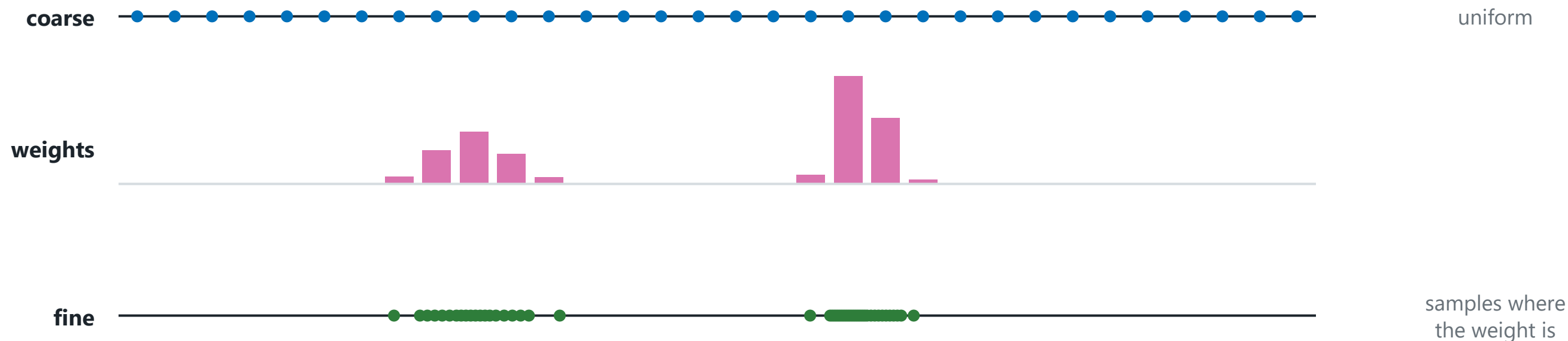
Part 4 · Neural radiance fields

- Each surface point coloured by how many training photos show it.
- Green: seen often — the model is well constrained. Red: seen once or never.
- Behind the box and under objects, any method can only guess.



coverage map of the 14 training views, seen from above

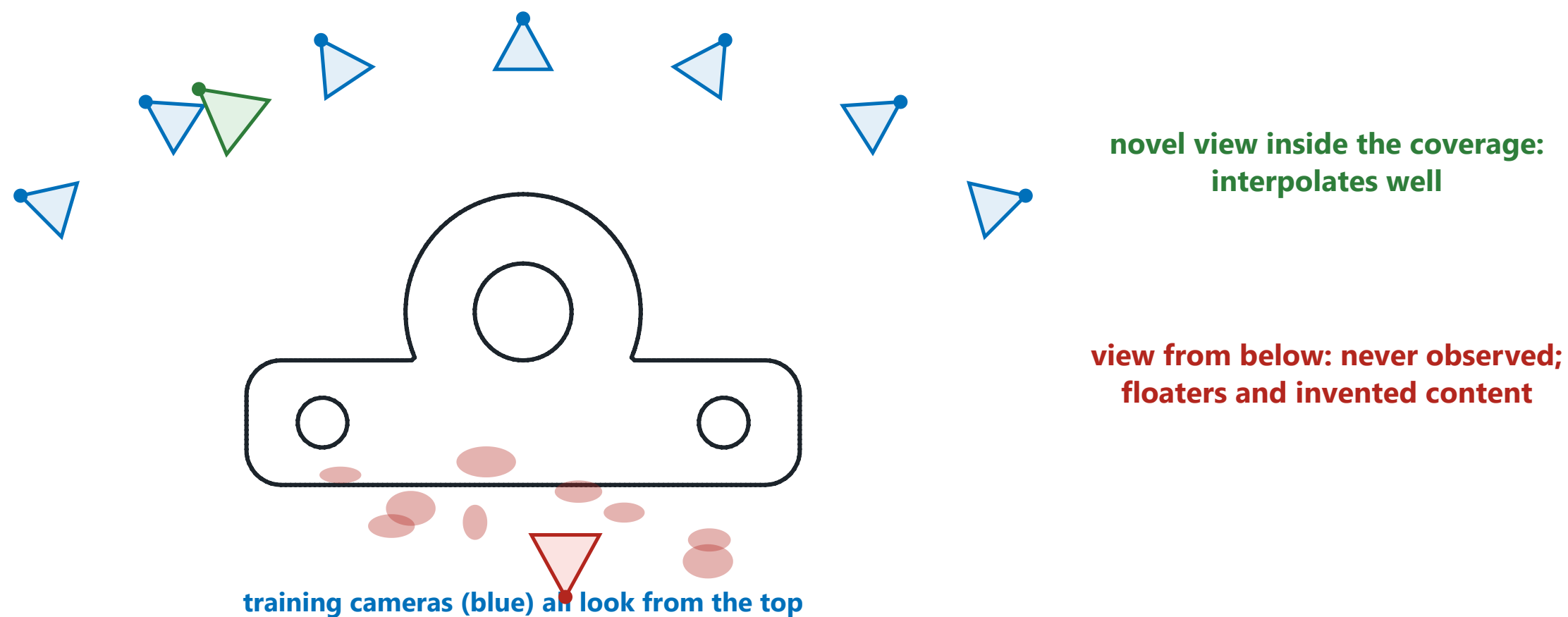
- Uniform samples waste most evaluations on empty space.
- A coarse pass estimates the weights along the ray.
- Fine samples are drawn where the weights are high — near the surface.



Where NeRF Works — and Where It Breaks

Part 4 · Neural radiance fields

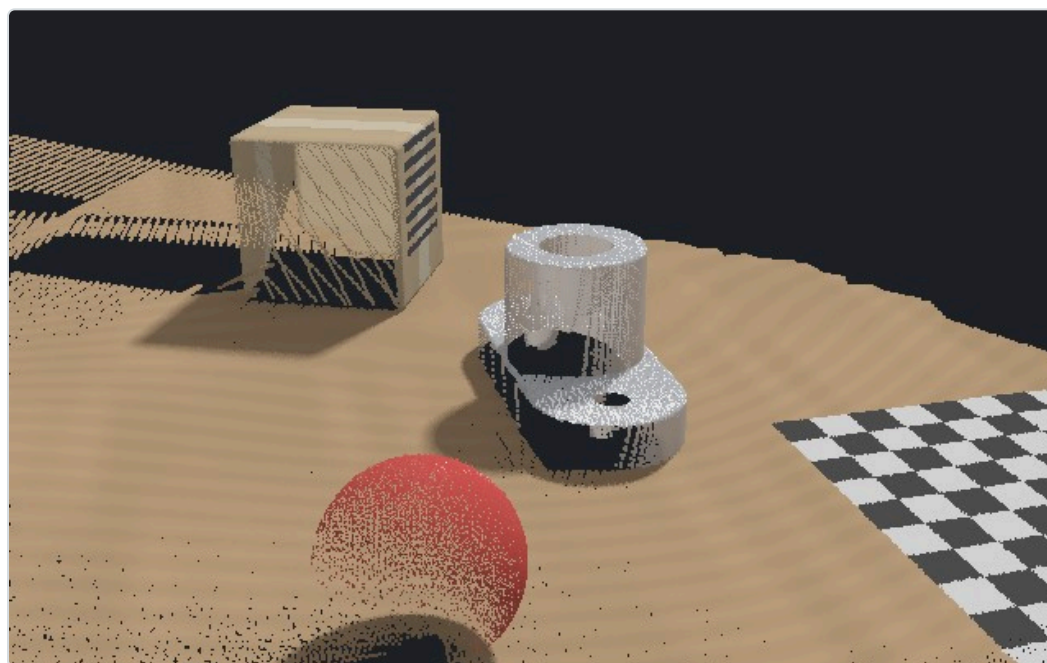
- New views between the training cameras are interpolated very well.
- Outside the camera coverage the model guesses: floaters and invented content.
- Few views or wrong poses make it worse.



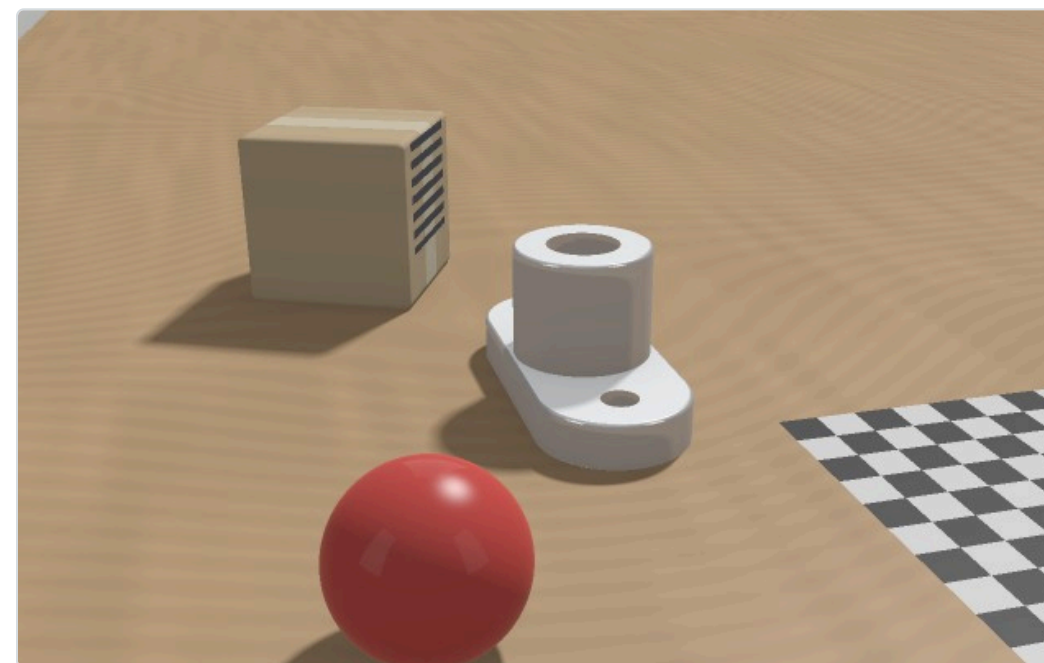
A New Viewpoint: What Is Known, What Is Not

Part 4 · Neural radiance fields

- The scene from a new side, outside the ring of training cameras.
- Left: everything the photos actually observed, re-projected. Right: the truth.
- A NeRF must fill the black gaps — that is where floaters and invented content appear.

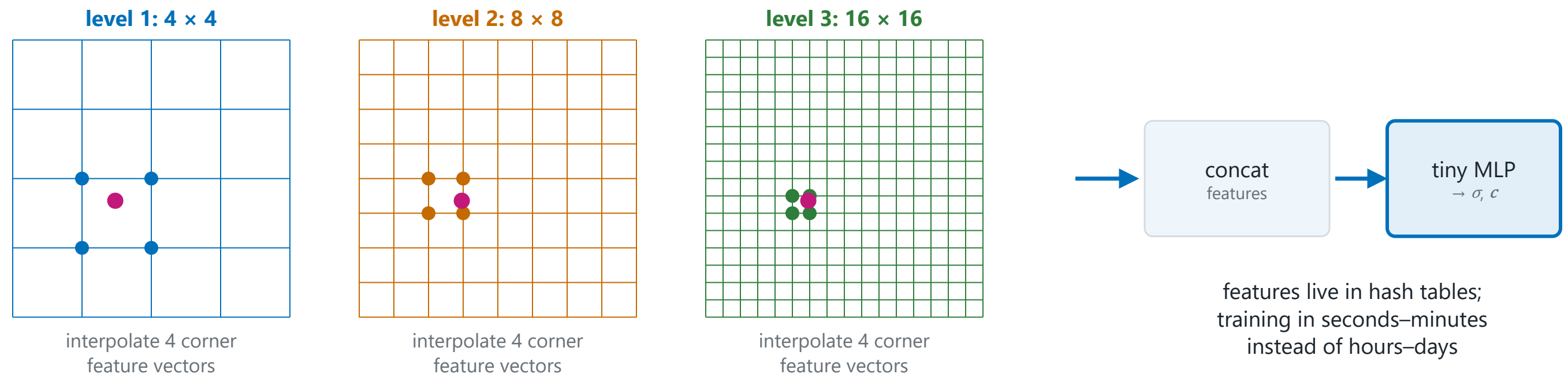


observed data from the new viewpoint



true view (never photographed)

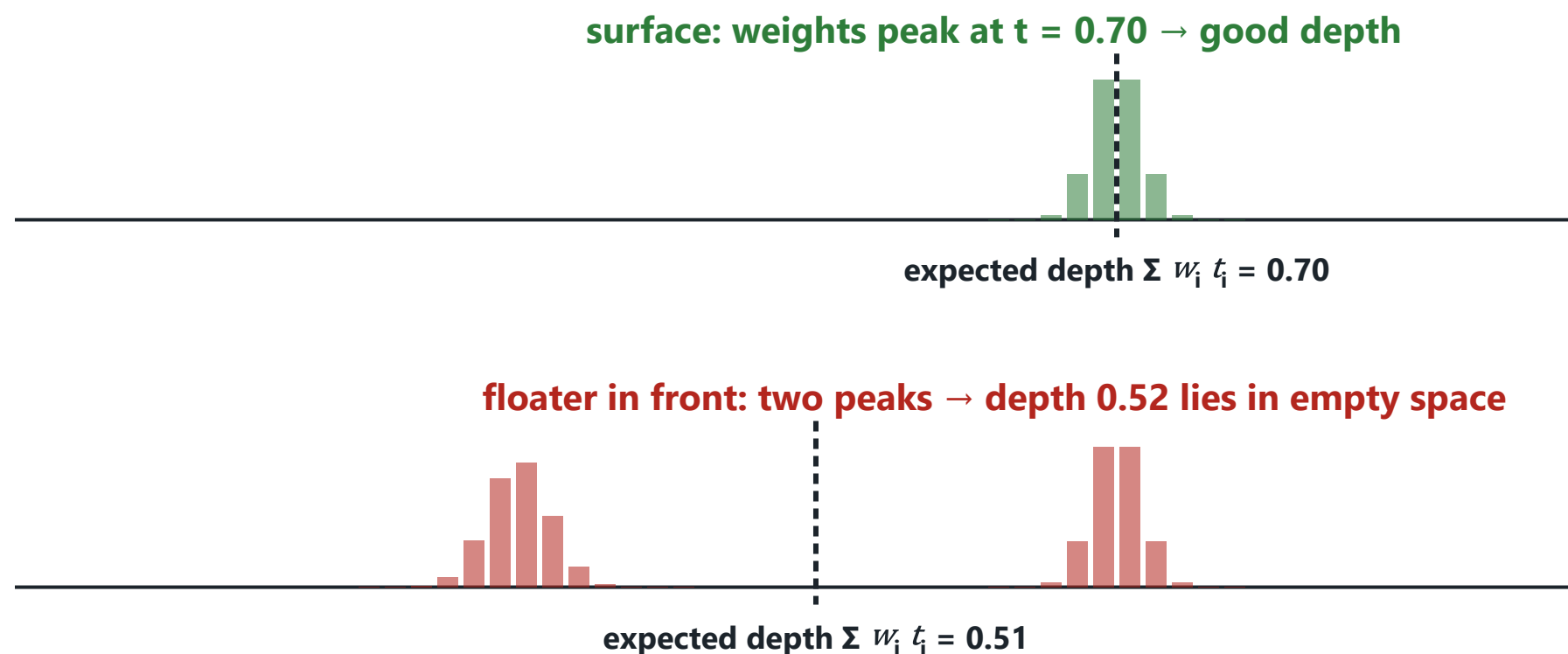
- The original NeRF took hours to days per scene.
- Instant-NGP stores learnable features in multi-resolution hash grids (Part 2).
- A tiny MLP reads the interpolated features: training in minutes.



Depth and Geometry from a NeRF

Part 4 · Neural radiance fields

- Expected depth along a ray: the weighted mean of sample depths.
- A mesh can be extracted from a density threshold — but the threshold is arbitrary.
- Floaters pull depth into empty space; SDF-based fields give cleaner surfaces.



depth and meshes can be extracted, but nothing forces the density to form a clean surface — SDF-based variants (NeuS, VolSDF) do

Your NeRF reaches a high PSNR on test views. Does that prove the part's dimensions are correct?

No. PSNR measures image agreement. Dimensions need geometric ground truth, calibration, errors in millimetres, and an uncertainty — and only where the part was observed.

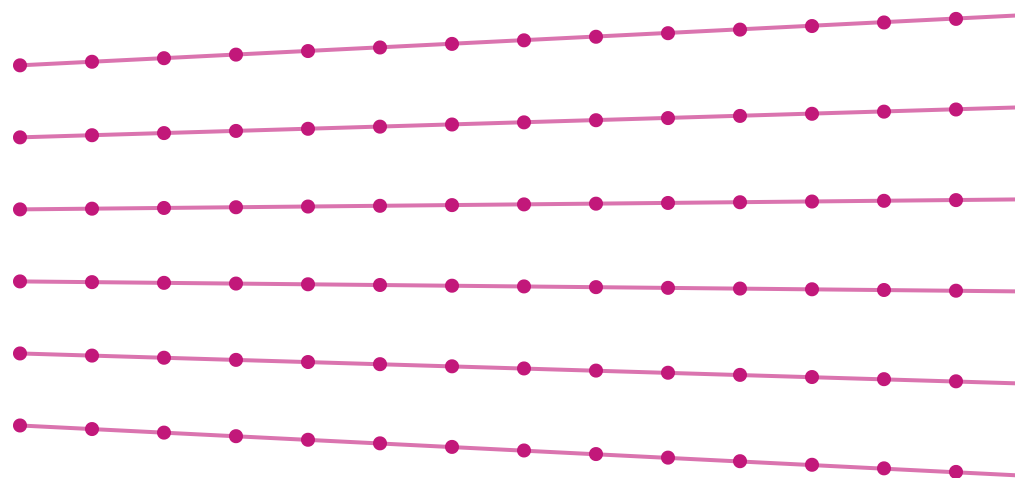


Why Gaussian Splatting?

Part 5 · Gaussian splatting

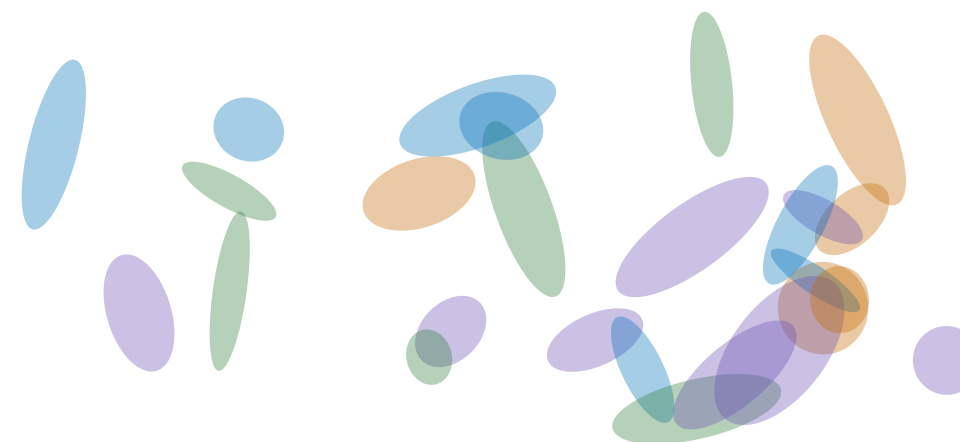
- NeRF evaluates a network hundreds of times per pixel.
- 3D Gaussian splatting (2023) stores explicit primitives and rasterizes them.
- Same training data (posed images), similar quality, real-time rendering.

NeRF: ask the network at every sample



1920 × 1080 pixels × ~200 samples
≈ 400 million network queries per frame

3DGS: project each primitive once

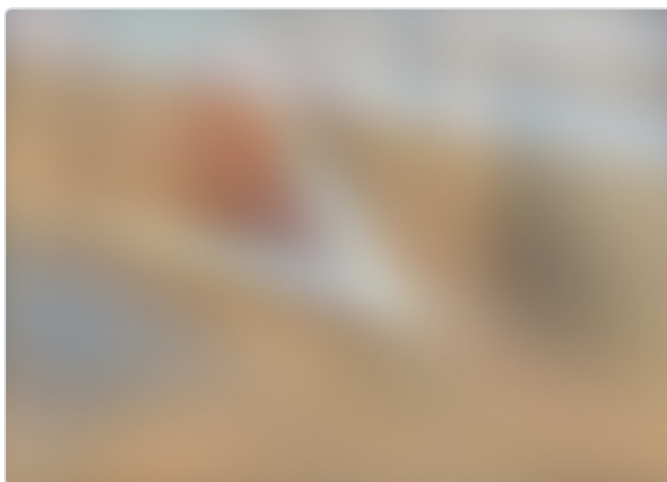


a few million Gaussians, sorted and blended
on the GPU: real time at 1080p

The Idea in 2D: Rebuild a Photo from Blobs

Part 5 · Gaussian splatting

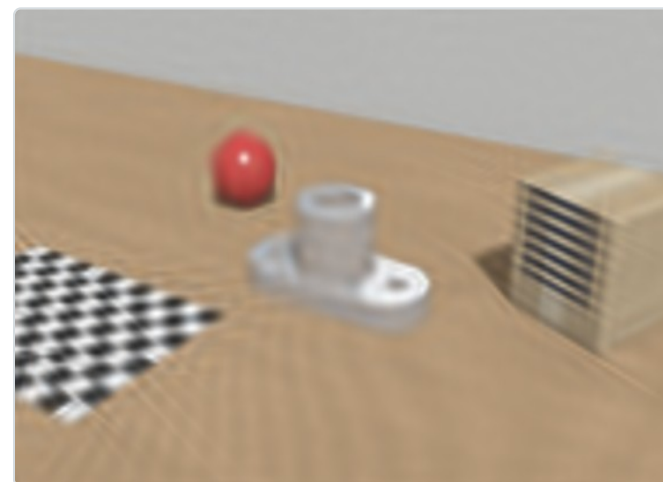
- Here we actually optimized coloured 2D Gaussians to reproduce the photo (gradient descent, as in lecture 1).
- Few blobs: a blurry impression. More blobs: detail appears where the image needs it.
- 3D Gaussian splatting does the same in 3D, with many photos at once.



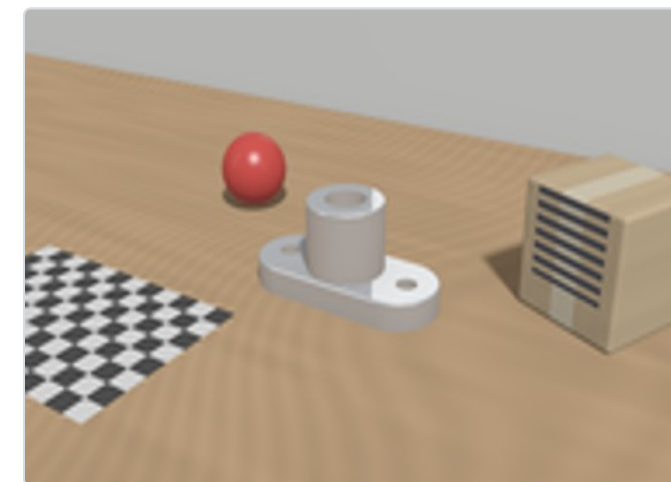
30 Gaussians · 22 dB



200 Gaussians · 25 dB



1200 Gaussians · 32 dB

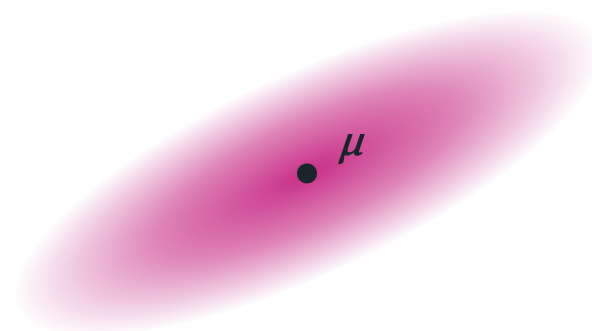


target photo

One 3D Gaussian

Part 5 · Gaussian splatting

- Centre μ , covariance Σ (size and orientation), opacity α , view-dependent colour.
- Σ is built from a rotation and three scales, so it stays valid during optimization.
- Try the sliders: anisotropy lets one Gaussian cover a thin edge.



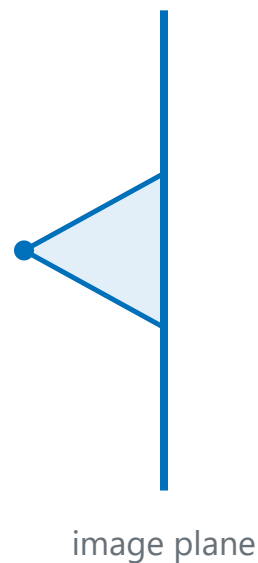
$$\Sigma = R S S^T R^T$$

$$\Sigma = [8.59 \ 3.54 ; 3.54 \ 2.65]$$

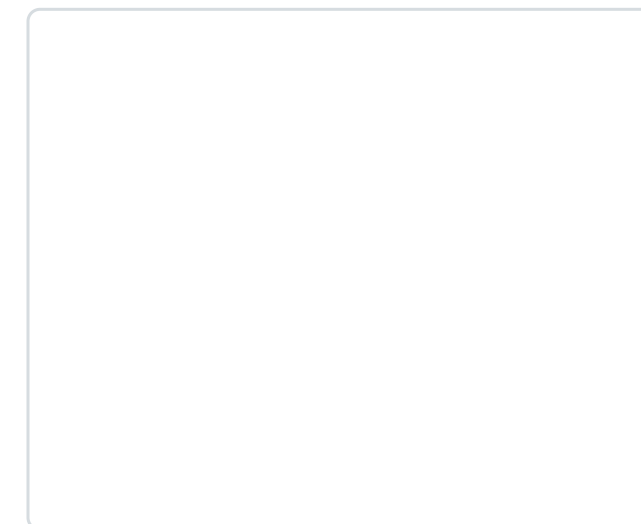
rotation R and scales S are optimized,
so Σ always stays a valid covariance

opacity $\alpha = 0.90$ · colour: spherical harmonics

- Each Gaussian is projected into the image as a 2D Gaussian (the splat).
- Its 2D covariance follows from the camera pose and the local linearization of the projection.
- Farther Gaussians make smaller splats.



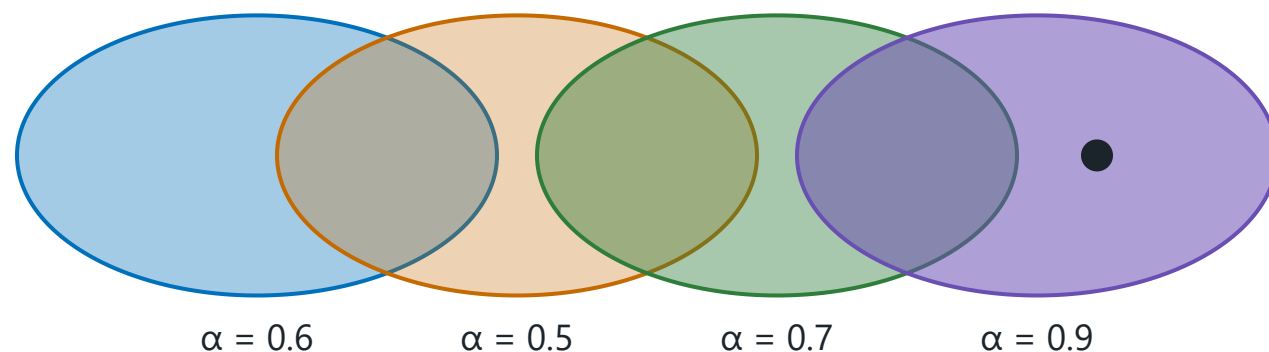
2D splat in the image



$$\Sigma = J W \Sigma W^T J^T \text{ (W: world} \rightarrow \text{camera, J: Jacobian of the projection)}$$

- Splats covering a pixel are sorted by depth.
- They are composited front to back with their opacities.
- It is exactly NeRF's rendering formula — with explicit primitives instead of samples.

splats overlapping one pixel, sorted front to back



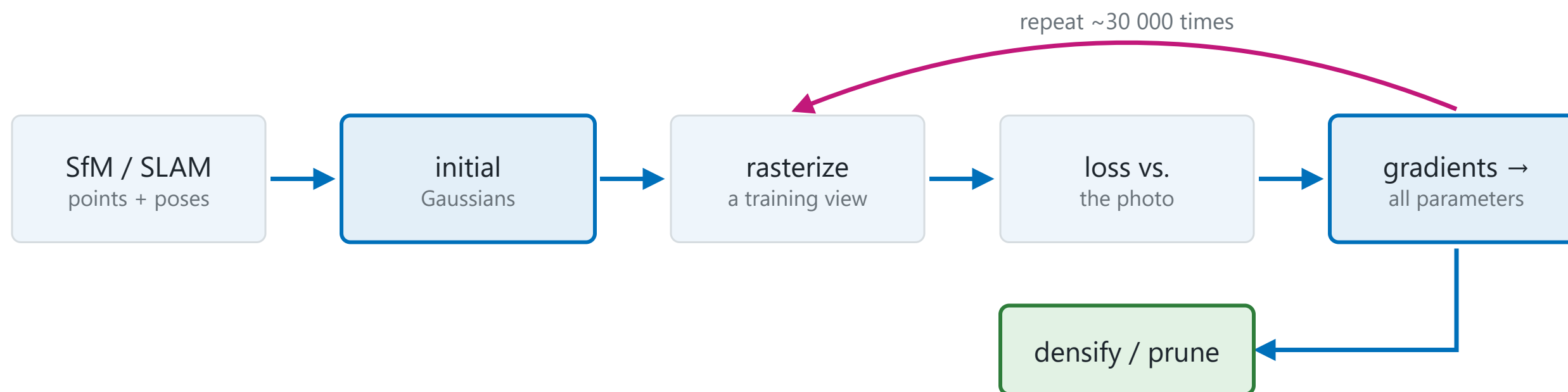
■	$T = 1.000 \quad w = T\alpha = 0.600$
■	$T = 0.400 \quad w = T\alpha = 0.200$
■	$T = 0.200 \quad w = T\alpha = 0.140$
■	$T = 0.060 \quad w = T\alpha = 0.054$

$C = \sum c_i \alpha_i \prod_{j < i} (1 - \alpha_j)$
— the same formula as NeRF's volume rendering

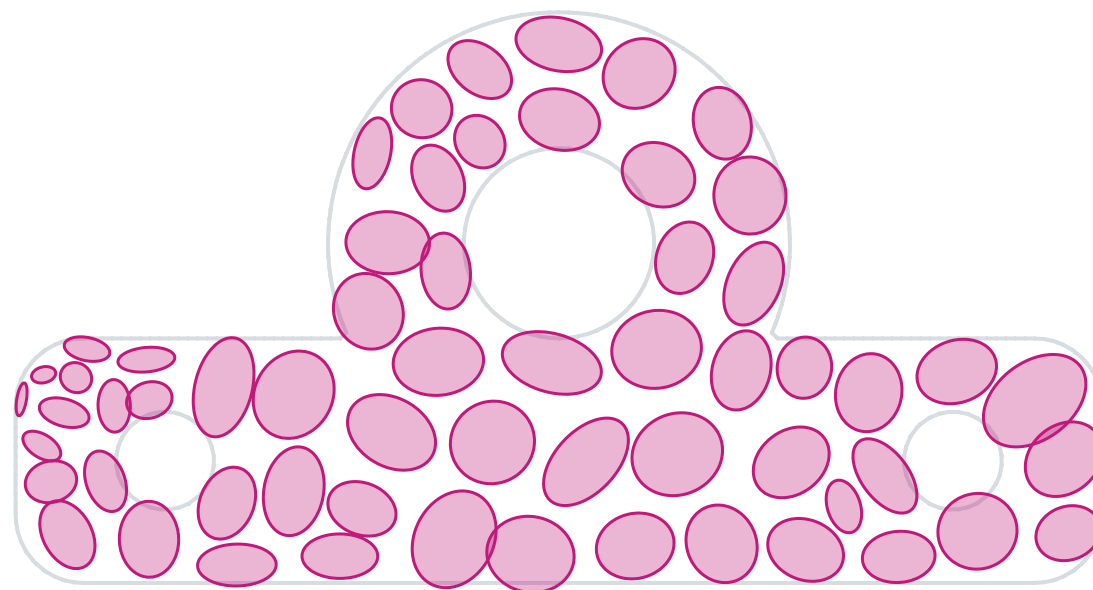
The Optimization Loop

Part 5 · Gaussian splatting

- Start from the sparse SfM/SLAM points: one Gaussian per point.
- Render a training view, compare with the photo, update every parameter by gradient descent.
- Periodically add and remove Gaussians.



- Where the image error stays high, small Gaussians are cloned and large ones split.
- Nearly transparent Gaussians are pruned.
- The number of primitives adapts to the detail of the scene (often millions).



prune nearly transparent ones; detail where needed

60 primitives

illustration on the 2D cross-section

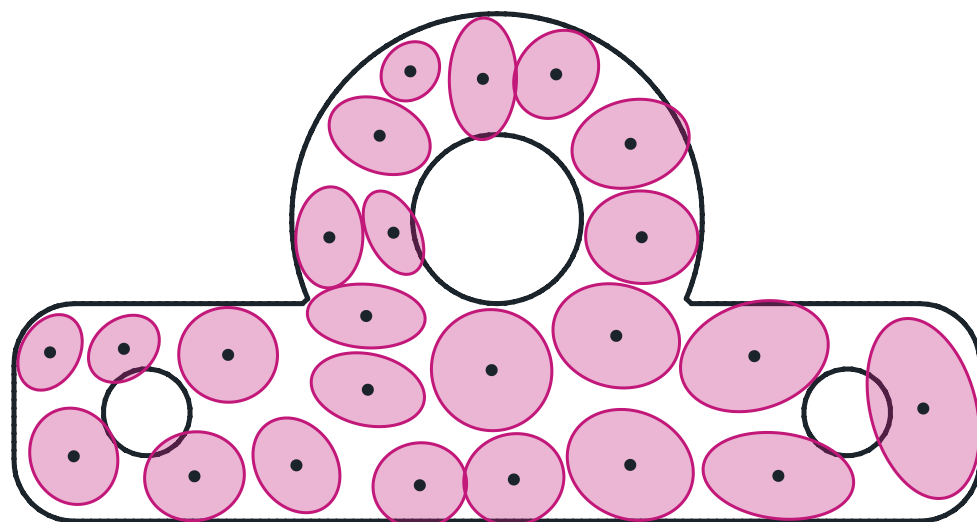
3DGS versus NeRF

Part 5 · Gaussian splatting

	NeRF (+ hash grids)	3D Gaussian splatting
scene stored as	network weights / feature grids	millions of explicit primitives
rendering	ray marching, many queries	rasterization, real time
memory	small	large (hundreds of MB)
editing	hard	easy: move, delete primitives
geometry	density threshold or SDF variant	needs surface-aligned variant
both	need calibrated, posed images and good coverage; optimize image agreement	

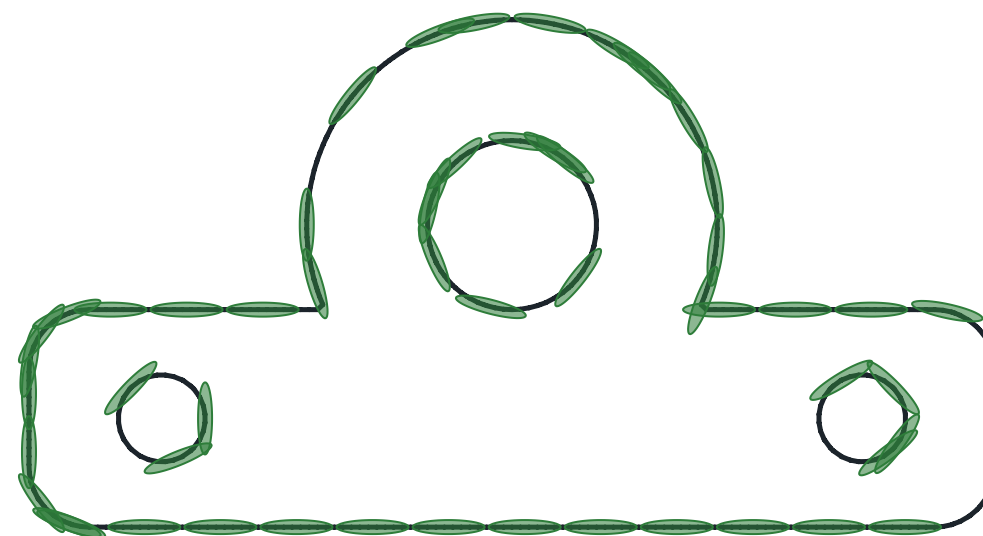
- Gaussian centres are not surface samples; blobs can explain images without a clean surface.
- Surface-aligned variants (2DGS, SuGaR) flatten the Gaussians onto the surface.
- Then depth and meshes can be extracted — and must still be validated.

3DGS: blobs explain the images



centres lie inside the object, not on its surface

surface-aligned (2DGS, SuGaR, ...)



flat discs on the surface: geometry can be extracted

Can you export the Gaussian centres as a point cloud and measure the part on it?

Not directly: centres sit inside the object and spread with the primitives' size. Use a surface-aligned variant or extract a surface, then compare with ground truth.

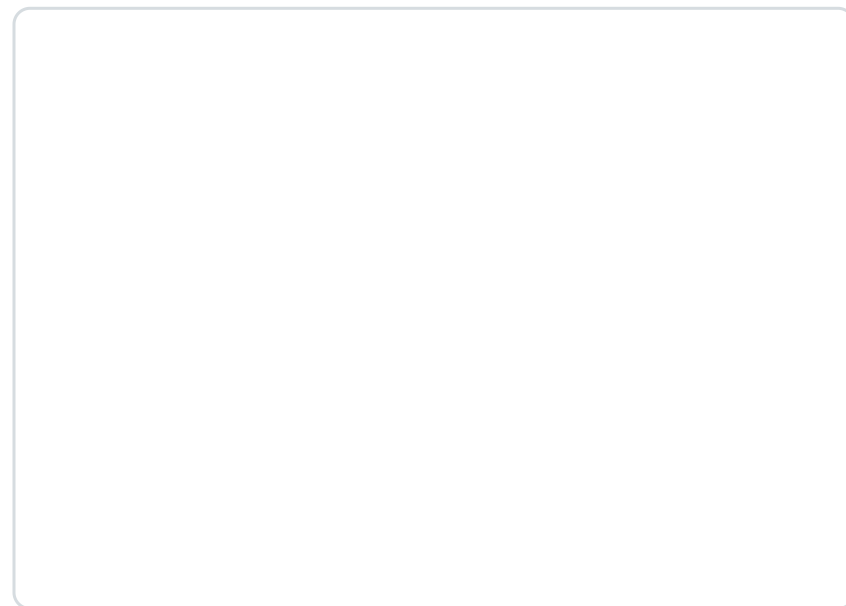


Who Moved: the Camera or the Scene?

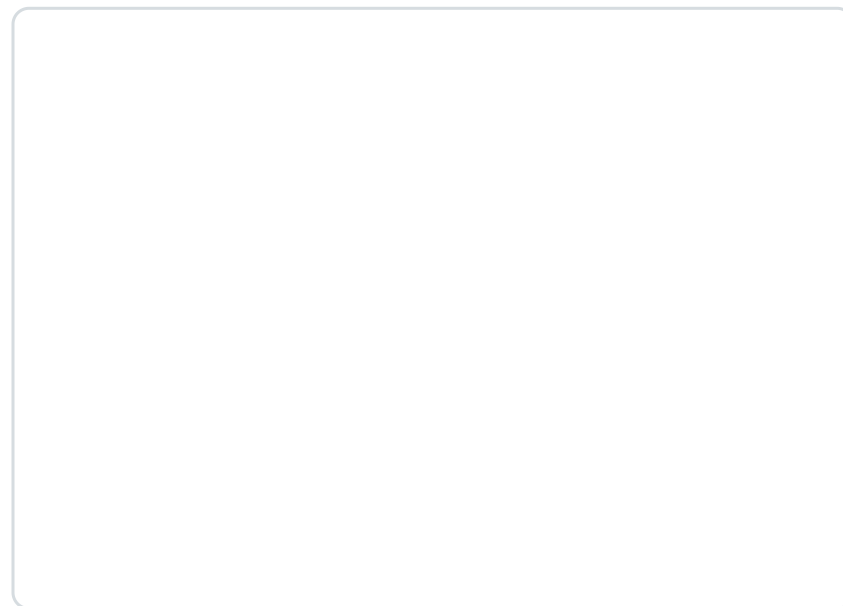
Part 6 · Dynamic scenes

- A dynamic scene is a function of position *and* time.
- From images alone, camera motion and object motion can explain the same change.
- Each time instant is usually seen from only a few views.

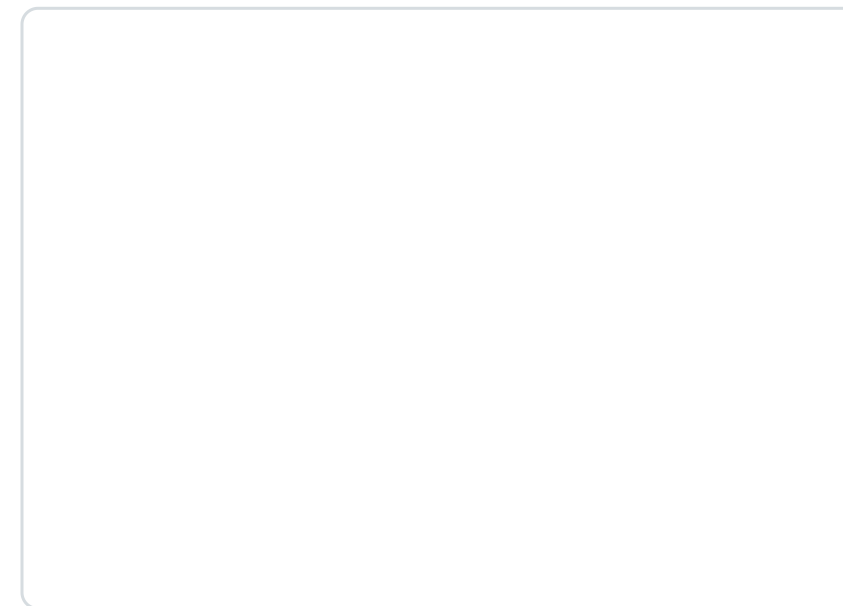
camera moves, object static



camera static, object moves



what both cameras record

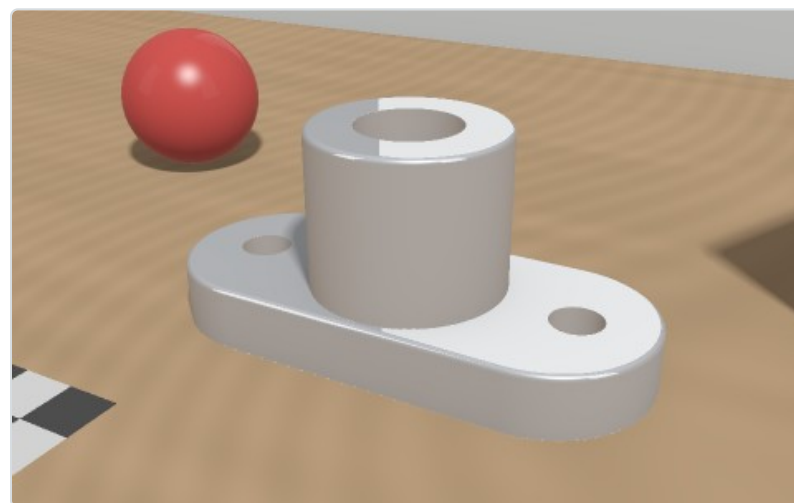


from the images alone both explanations fit: 4D methods must separate camera motion from scene motion

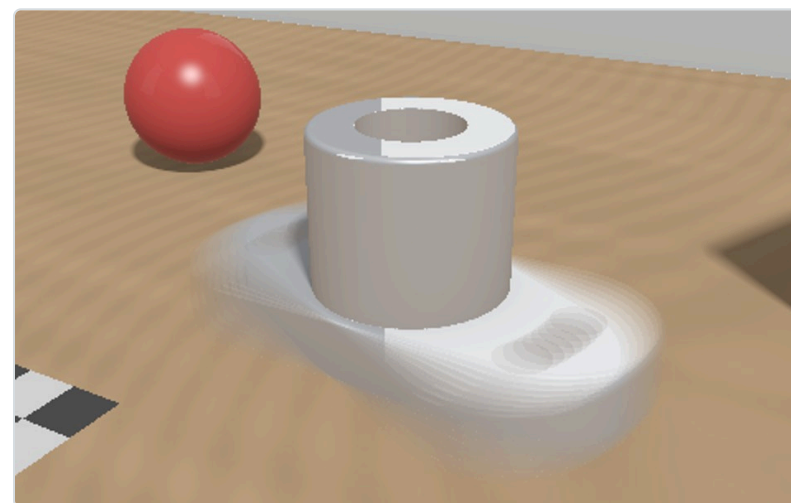
Moving Objects Break Simple Cameras

Part 6 · Dynamic scenes

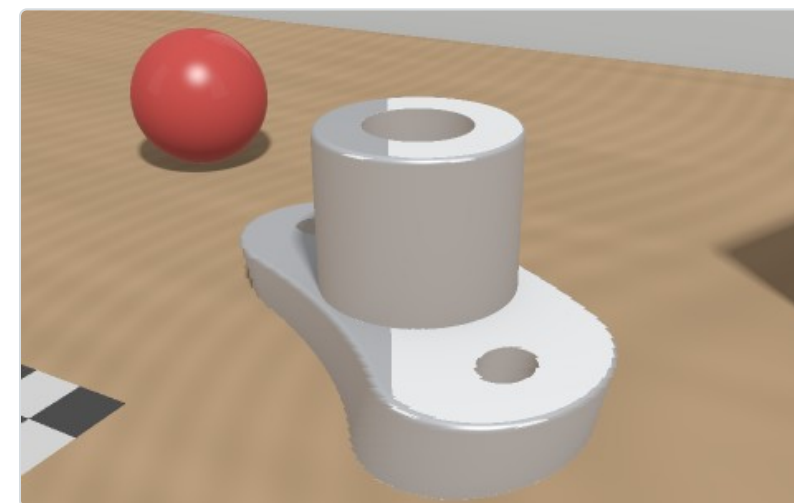
- The flange rotates during the exposure. Same scene, three cameras.
- Long exposure smears it; a rolling-shutter camera reads rows at different times and bends it.
- A 3D method fed with these images reconstructs the smear or the bend as if it were real.



global shutter, short exposure



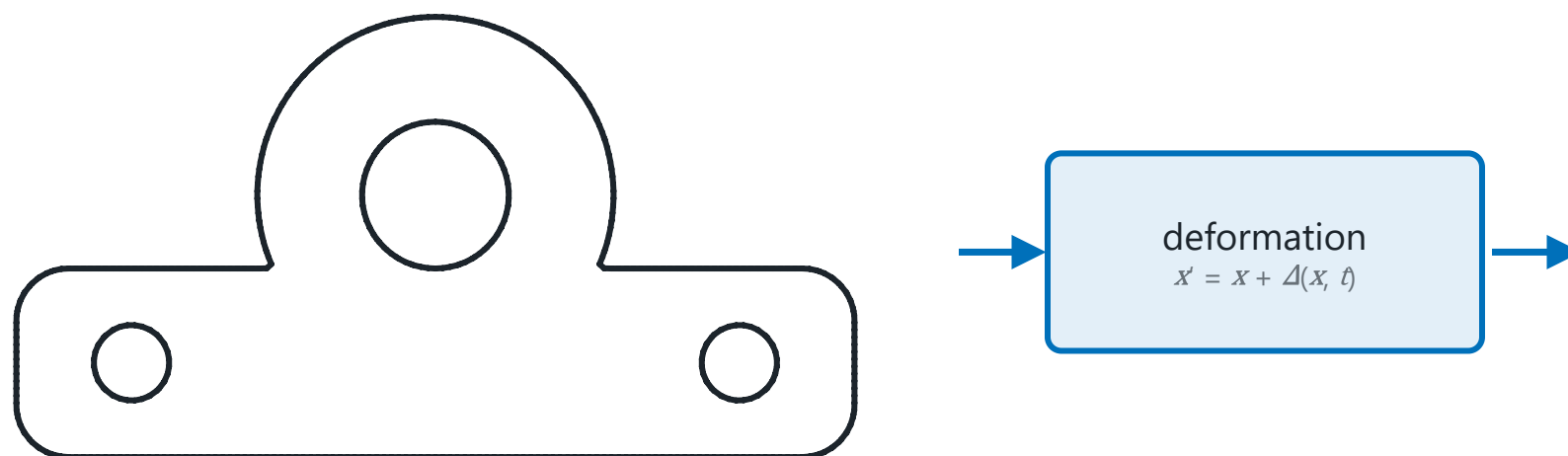
long exposure: motion blur



rolling shutter: skewed shape

- Store the scene once, in a canonical (reference) configuration.
- A deformation field moves every point to its position at time t .
- Structure is shared over time; topology changes (cutting, opening) remain hard.

canonical shape (shared over time)



a NeRF or a set of Gaussians lives in the canonical space; a small network moves it to time t

4D Gaussian Splatting

Part 6 · Dynamic scenes

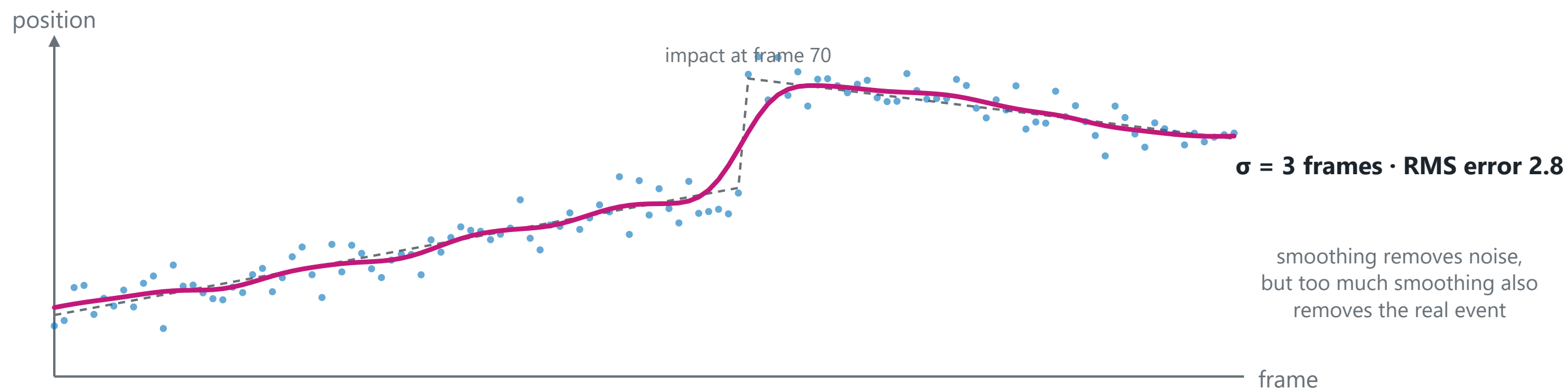
- Each Gaussian gets a time-dependent position, and possibly shape and colour.
- Motion is modelled by a small network, by trajectories, or by a time basis.
- Rendering at any time stays a fast splat.

each Gaussian gets a
time-dependent position
(and possibly shape, colour):

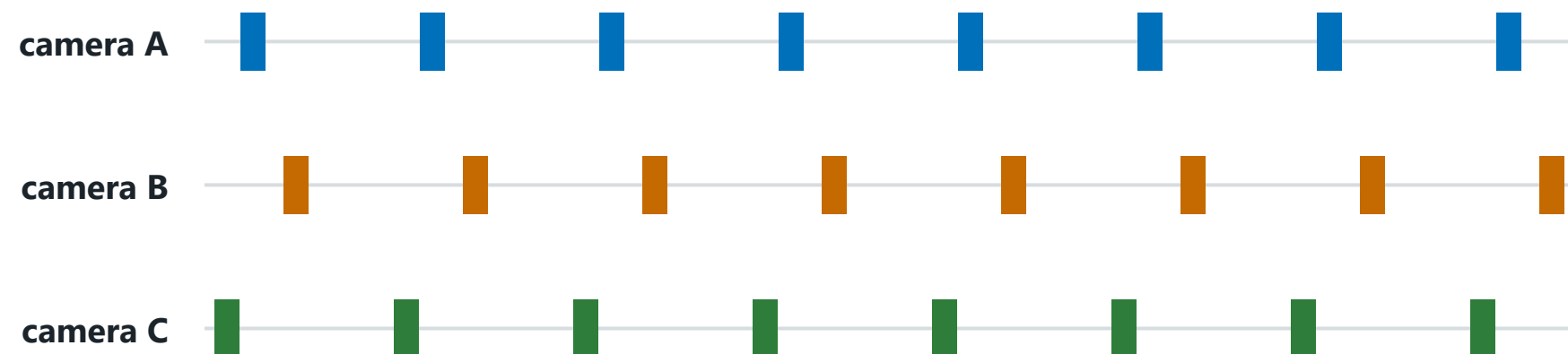
$$\mu(t), \Sigma(t)$$

rendering stays a fast splat
of the Gaussians at time t

- Few observations per instant make motion estimates noisy.
- Smoothness over time suppresses jitter and flicker.
- Too much smoothing removes real events: contacts, impacts, sudden stops.



- Several synchronized cameras are worth more than a better model.
- An unsynchronized camera sees moving objects at the wrong place.
- Short exposure limits motion blur; coverage in space and time limits guessing.



offset 8 ms × speed 1.5 m/s

= 12 mm apparent displacement

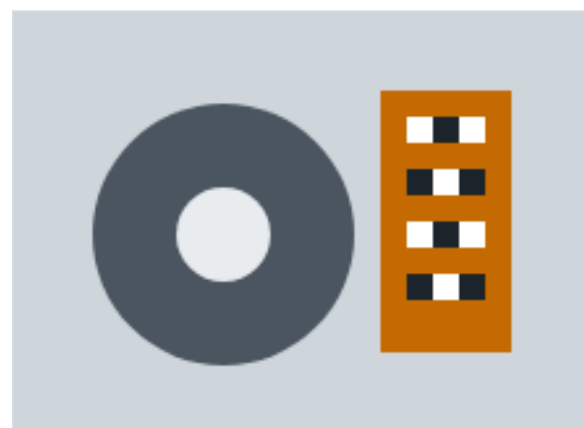
unsynchronized views see a moving
object at different places: the
reconstruction blurs or doubles it

You want a 4D model of a robot arm moving at 2 m/s with 5 mm accuracy. What do you require from the camera setup?

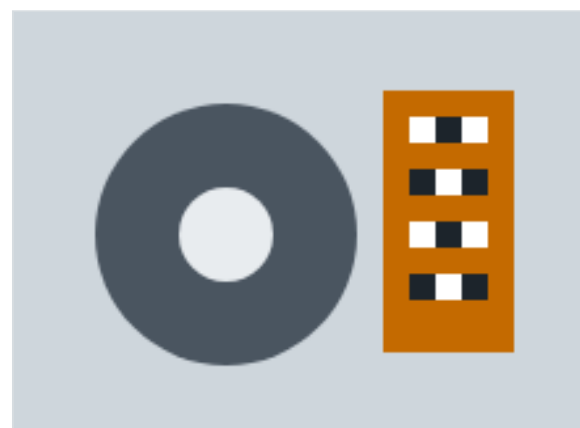
Hardware-synchronized cameras (offset well below 2.5 ms), short exposure, enough views around the arm, and calibrated poses — before choosing any 4D method.



- PSNR measures pixel error, SSIM local structure, LPIPS perceptual similarity (a network).
- Always evaluate on cameras held out from training.
- Pixel metrics are very sensitive to tiny misalignment and exposure changes.



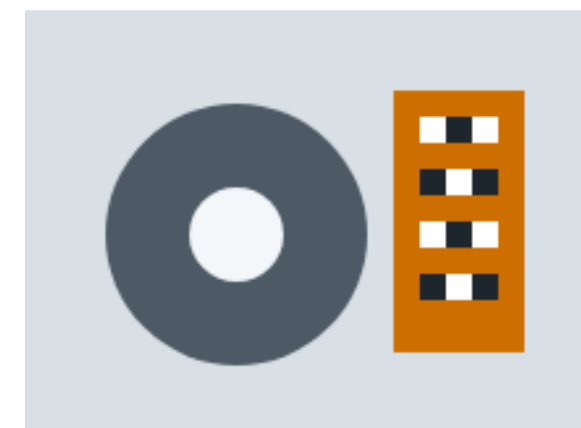
reference
PSNR ∞



shifted by 1 px
PSNR 22.7 dB



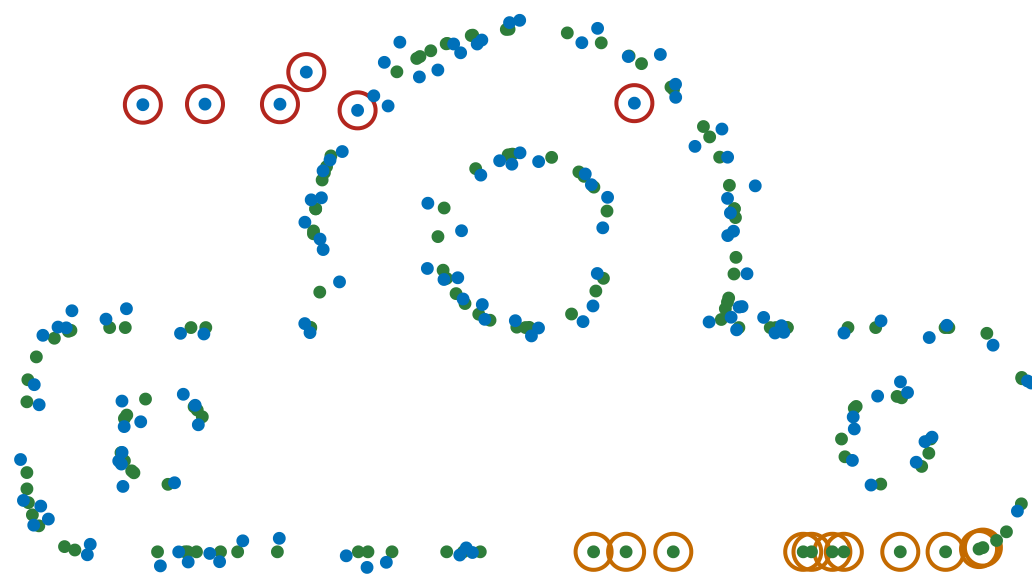
slightly blurred
PSNR 26.6 dB



5 % brighter
PSNR 29.5 dB

PSNR = $10 \log_{10}(1 / \text{MSE})$, computed live on these images. A 1-pixel shift costs more than visible blur: pixel metrics punish misalignment, not perceived quality.

- Accuracy: how far reconstructed points are from the truth (false surface).
- Completeness: how much of the true surface was reconstructed (missing surface).
- Chamfer distance averages both; the F-score reports both at a tolerance τ .



● ground truth (scanner)

● reconstruction

accuracy (rec $\rightarrow$ GT): 1.2 mm

completeness (GT $\rightarrow$ rec): 1.4 mm

Chamfer = mean of both: 1.3 mm

at $\tau = 3$ mm: precision 96 %, recall 92 %

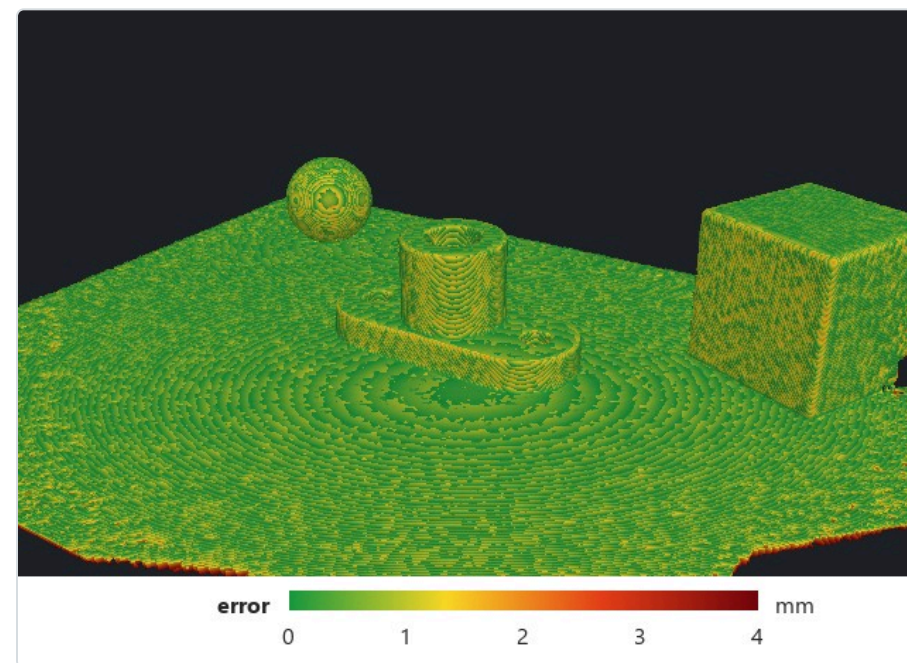
F-score 94 %

red: false surface · orange: missing surface

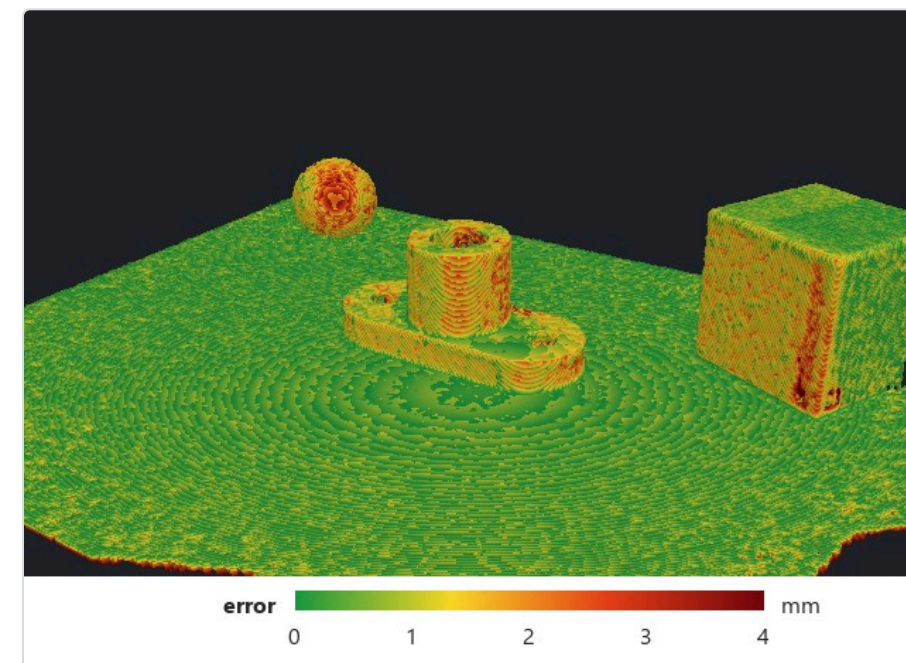
A Small Pose Error, a Visible Shape Error

Part 7 · Evaluation and practice

- The same 14 depth maps fused twice; in the second run, three camera poses are off by about 4° .
- The table (seen the same way by all cameras) stays green; edges and curved objects turn red.
- Such maps are how calibration and pose problems are found in practice.

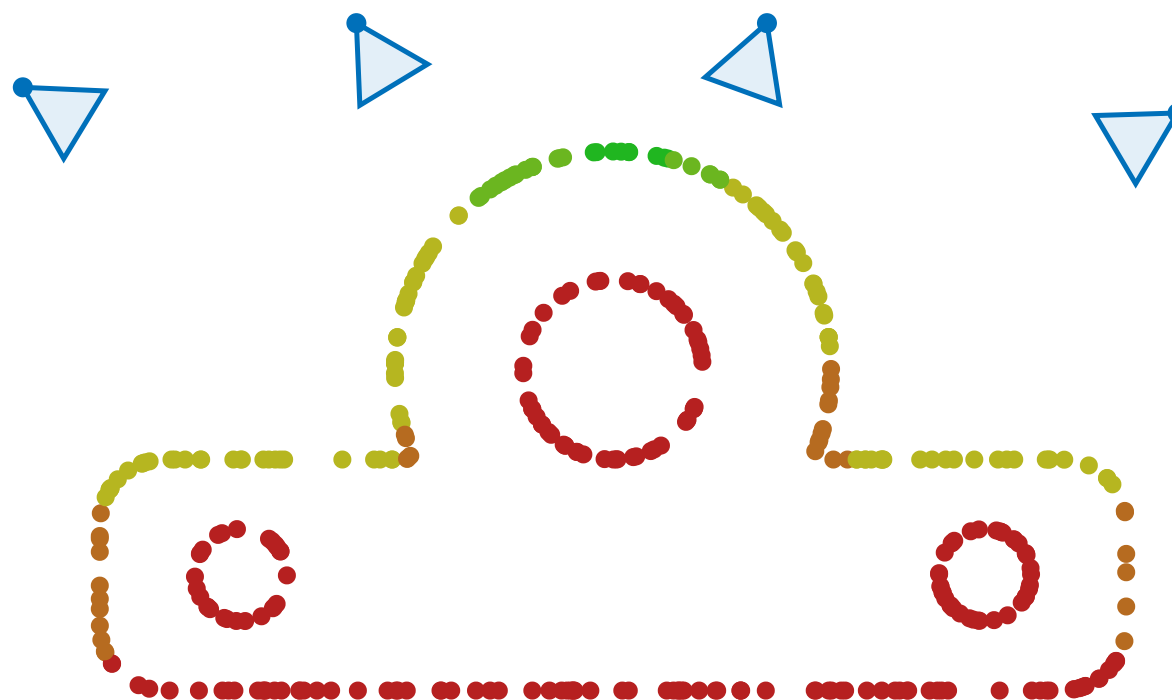


correct poses (0–4 mm)



3 poses off by $\approx 4^\circ$

- Aleatoric uncertainty: sensor noise and ambiguity in what was observed.
- Epistemic uncertainty: what the data never constrained — the model's guess.
- A reconstruction without a confidence map is incomplete for decisions.



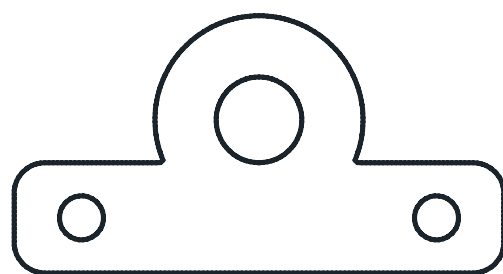
colour = how many cameras
saw this part of the surface

green: well observed
red: never observed → the
model's guess (epistemic)

aleatoric: noise and ambiguity
even where it was observed

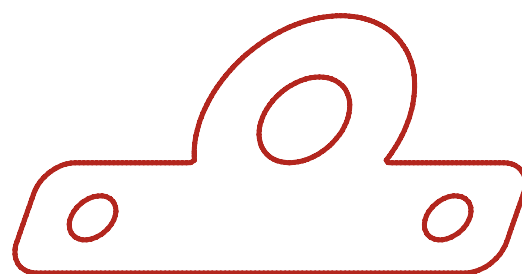
- Each failure has a physical cause, not only a model cause.
- Most can be detected by checking geometry against a reference.
- The last one is the most dangerous: the result looks plausible.

shiny / transparent



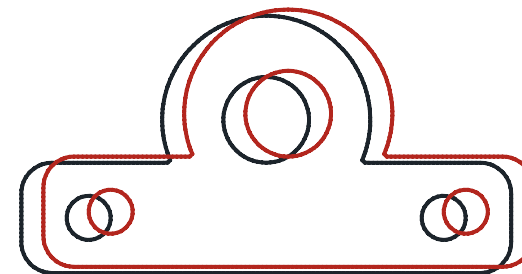
reflections become
fake geometry

rolling shutter, motion



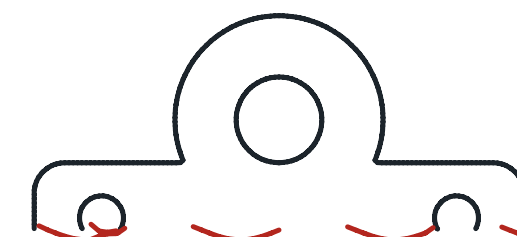
shapes skew and smear

wrong camera poses



surfaces double or blur

unobserved regions



plausible but invented

- Dense 3D models connect perception with simulation, robotics, and inspection.
- The required quality depends on the application.
- Decide the quality measure before choosing the method.

DIGITAL TWIN

inspect a part or a line from any view,
simulate camera placements

ROBOT WORKSPACE

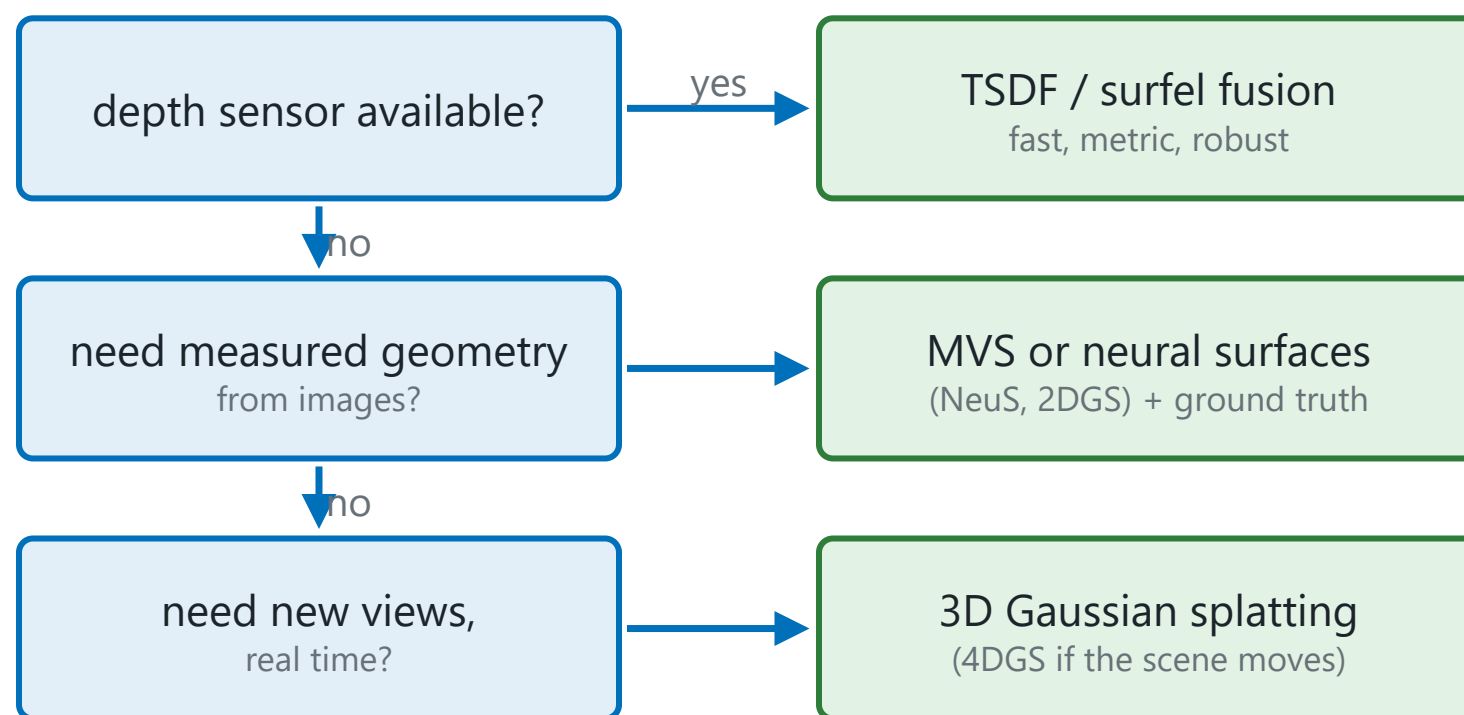
dense geometry for grasping and
collision checking

PROCESS MONITORING

4D models of people, machines,
material flow

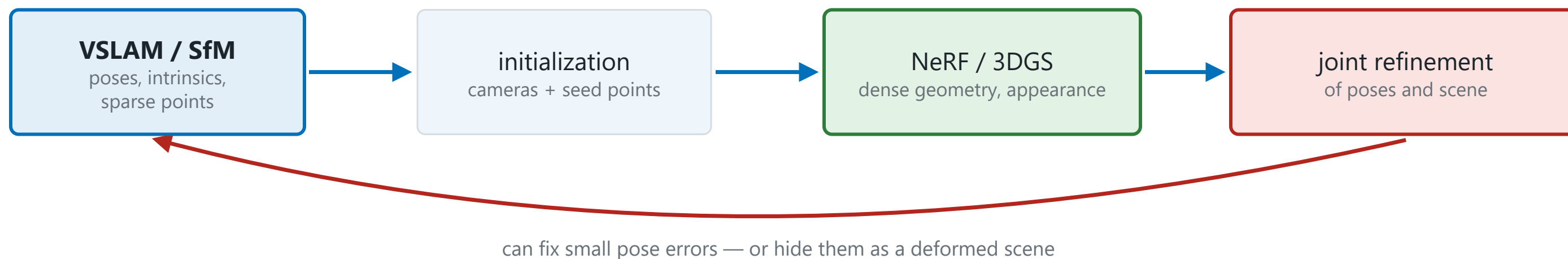
each needs a different representation and a different quality measure

- Start from the data you have and the output you need.
- The simplest pipeline that meets the requirement wins.
- The checks at the end are not optional.

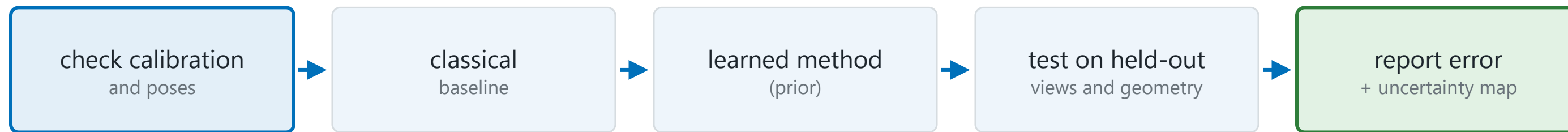


in every case: check calibration and poses first,
compare with a classical baseline, evaluate
geometry on held-out data

- VSLAM provides poses, intrinsics, and seed points for NeRF and 3DGS.
- Refining poses together with the scene can remove small errors.
- It can also absorb a pose error into a deformed scene — check the poses separately.



- Verify calibration and poses before anything learned.
- Keep a classical baseline to compare against.
- Report geometric error and uncertainty on data the method has not seen.



base operational decisions on measured geometry and its uncertainty, not on how good the rendering looks

How do you show that a reconstruction is correct, not just a good-looking rendering?

Hold out views for image metrics, compare geometry with an independent reference in millimetres (accuracy, completeness, F-score), and mark regions that were never observed.



What to Remember

Part 7 · Evaluation and practice

- The representation decides what can be measured and what it costs.
- NeRF and 3DGS optimize agreement with images, not automatically correct surfaces.
- Time adds ambiguity and strict capture requirements.

REPRESENTATION

decides what can be measured
and what it costs

LEARNED DEPTH

priors fill gaps, scale needs
a reference

NeRF / 3DGS

optimize image agreement,
not surfaces

EVALUATION

geometry in mm, held-out views,
uncertainty

What output does the next algorithm actually need?
Which part of the result is measured, and which is prior?
Where — and in which units — can we verify it?

