Lecture 16:

Real-time Dense 3D Reconstruction

Visual Computing Systems
CMU 15-769, Fall 2016

Last week:

- Large-scale (off-line) sparse 3D reconstruction
- From unstructured collections of millions of images

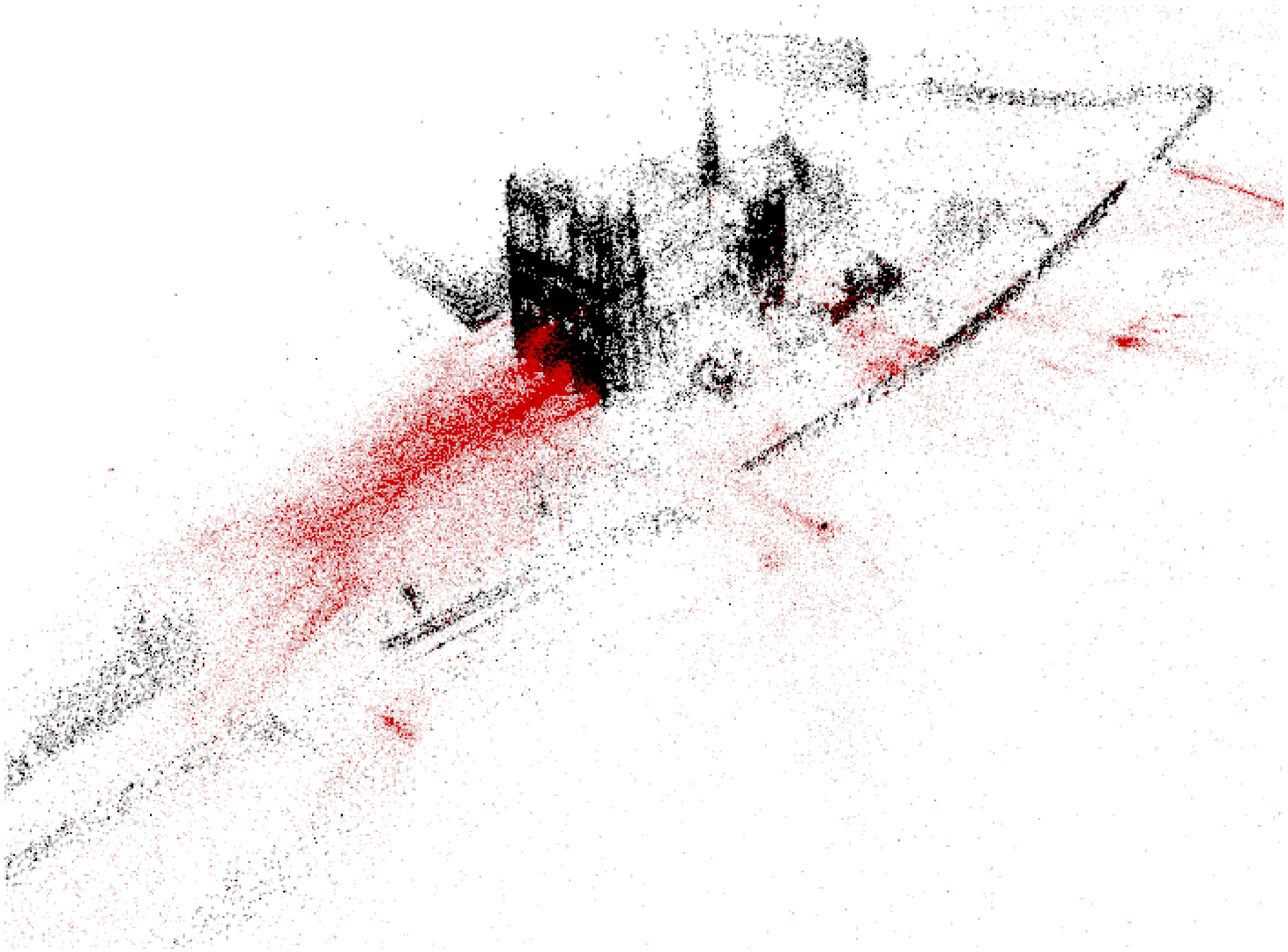


Image credit: Heinly 2015

Today: real-time, dense 3D reconstruction from a single RGBD capture session

Microsoft Kinect

Occipital's Structure Sensor

Stereolabs Zed

3D output

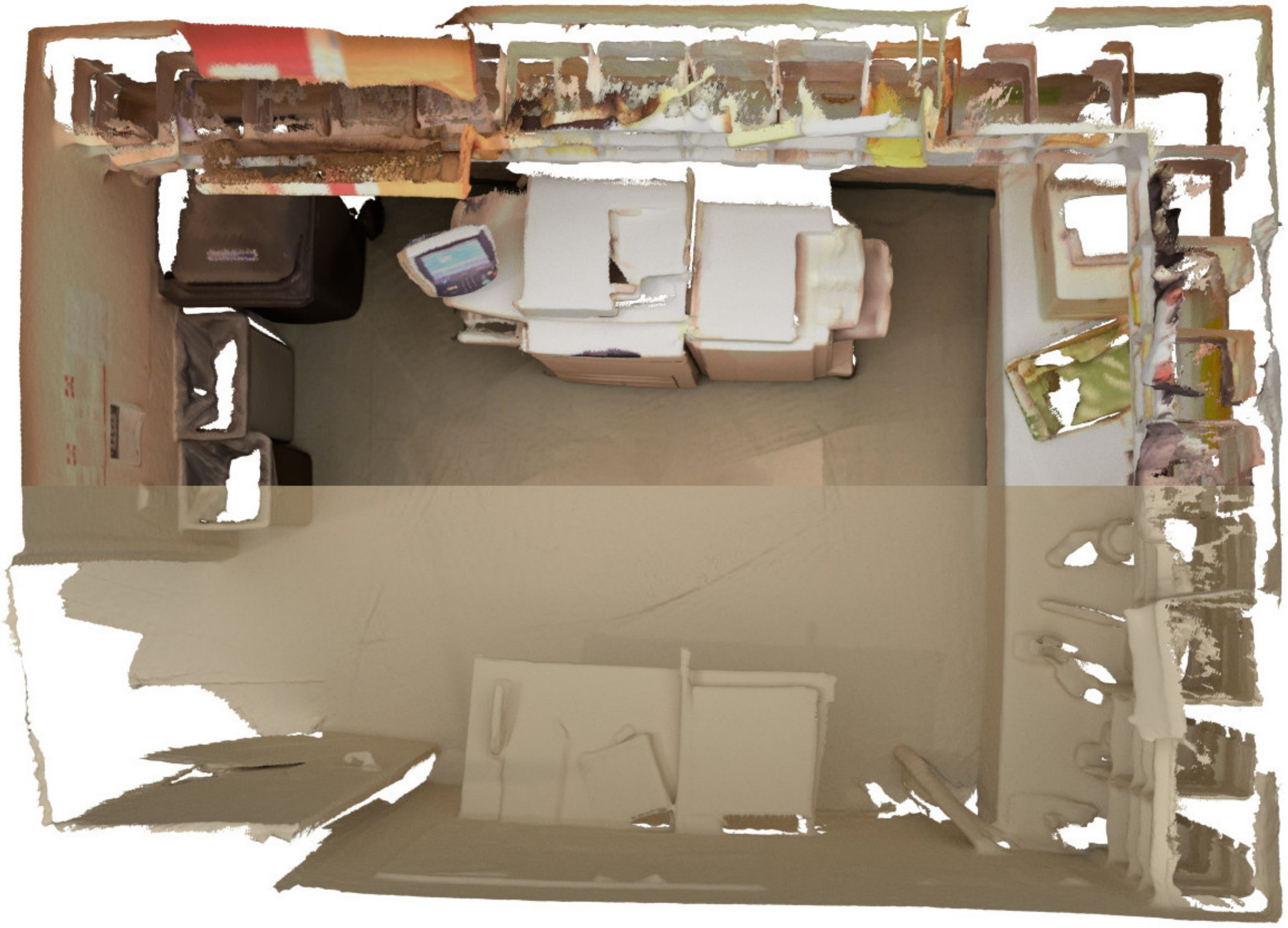


Image credit: Dai 2016

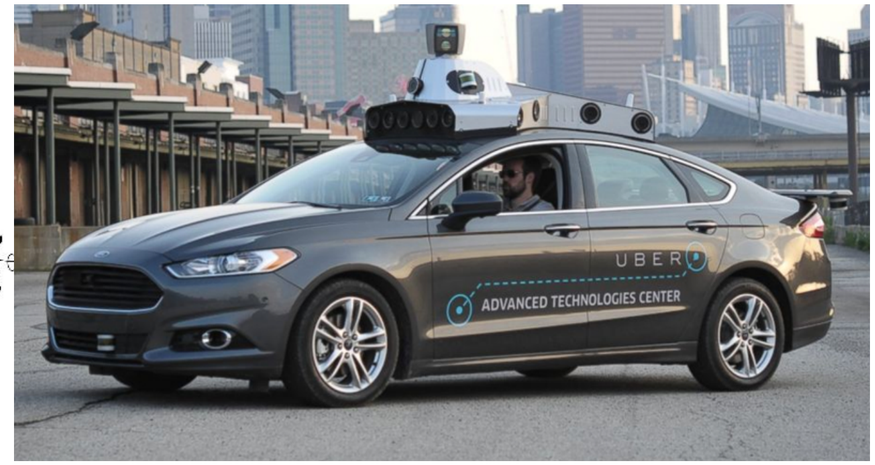
3D output

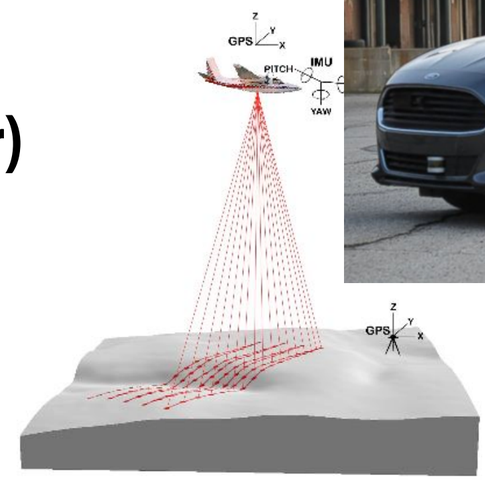
Image credit: Dai 2016

Aside: how a depth camera works

Depth from time-of-flight

- **Conventional LIDAR**
  - Laser beam scans scene (rotating mirror)
  - Low frame rate to capture entire scene





- **“Time-of-flight” (TOF) cameras**
  - No moving beam, capture entire image of scene with each light pulse
  - Special CMOS sensor records a depth image based on phase of arriving light (phase depends on distance at which light is reflected off scene object)
  - High frame rate
  - Formerly TOF cameras were low resolution, expensive…
  - TOF camera featured in XBox One depth sensor (today we will only talk about the original XBox 360 implementation)

Computing depth of scene point from two images

Assume two calibrated cameras are looking at the same scene

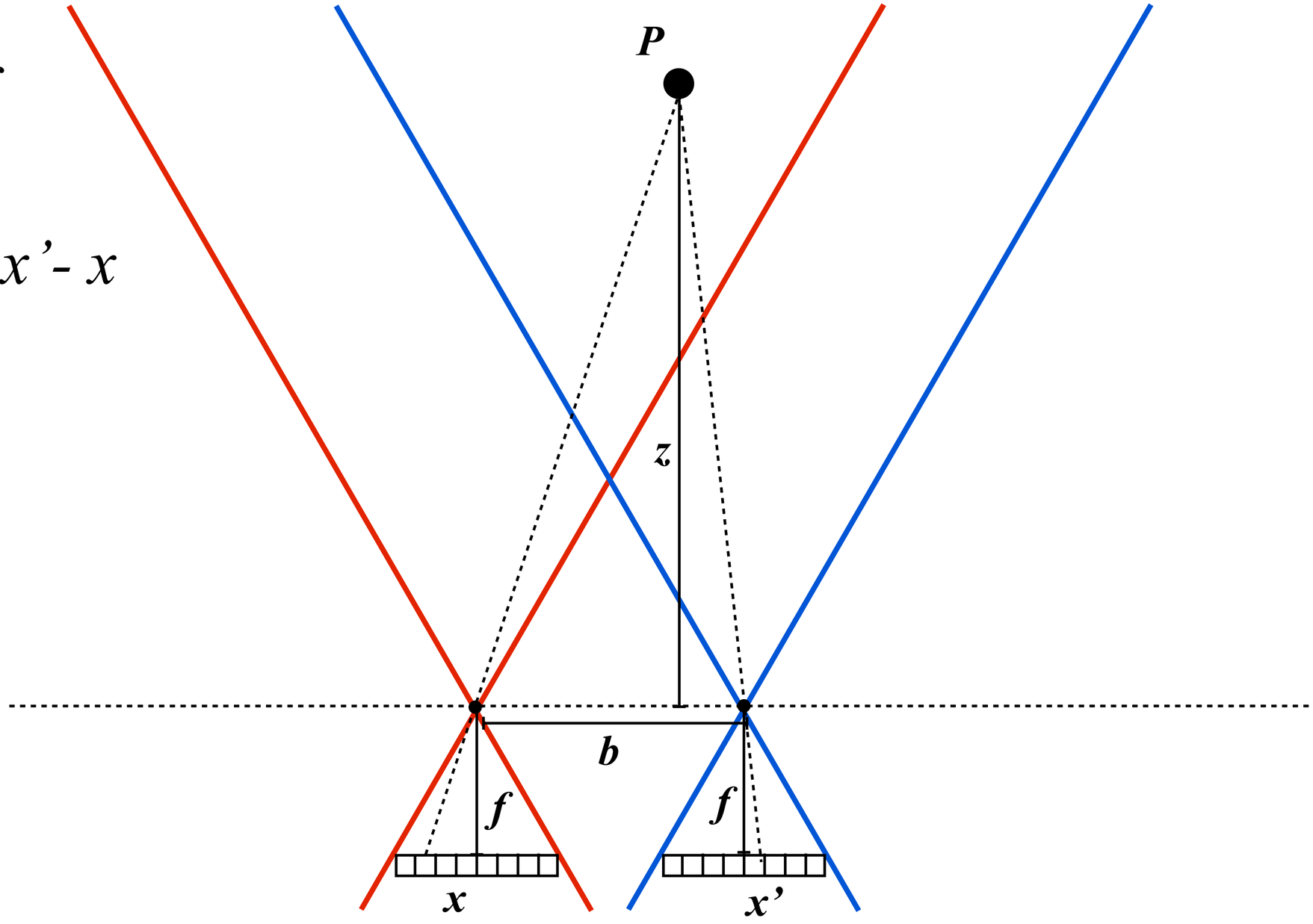
Binocular stereo 3D reconstruction of point *P*: depth from disparity

Focal length: f

Baseline: b

Disparity: $d = x' - x$

$$z = \frac{bf}{d}$$

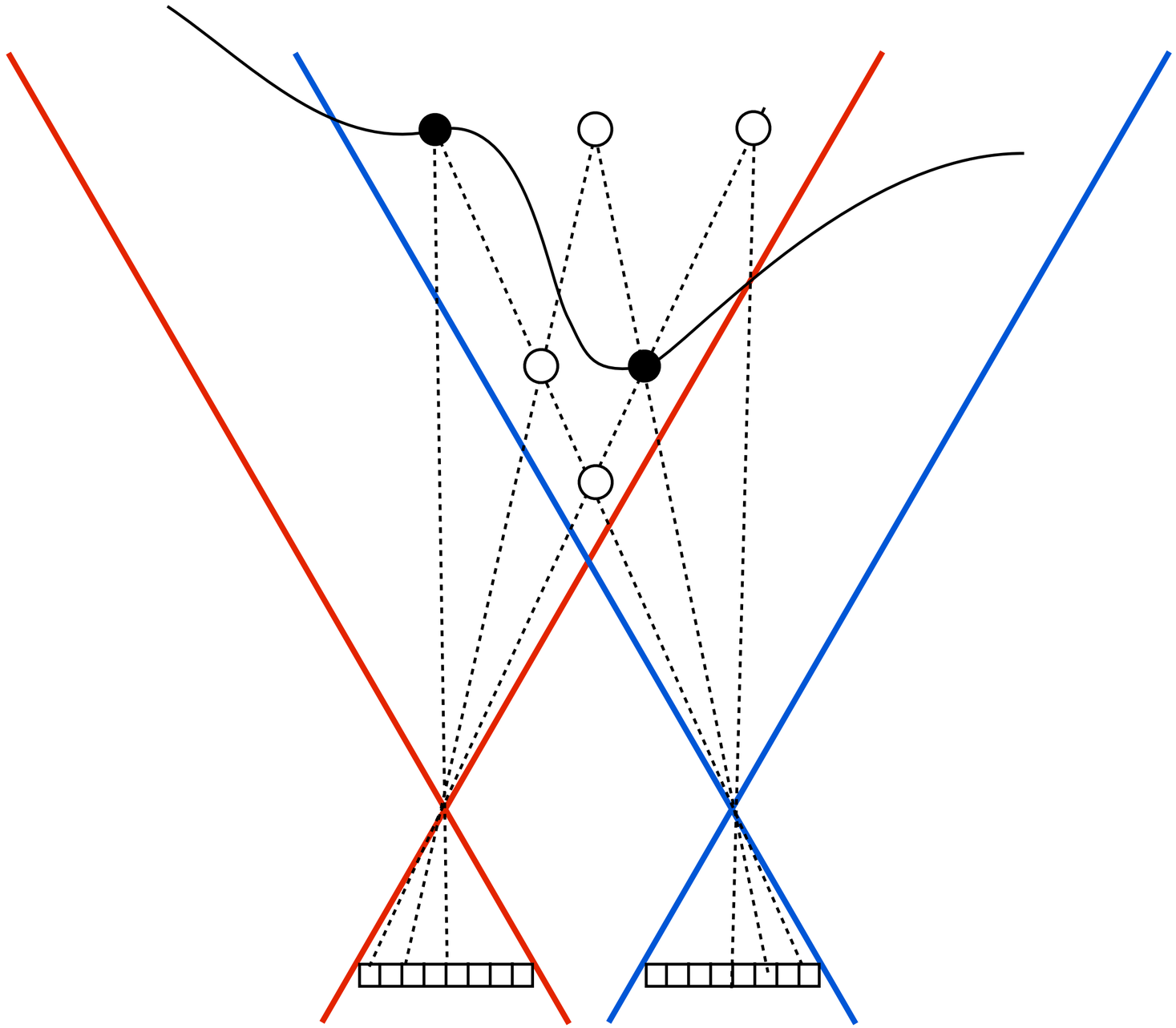


Simple reconstruction example: cameras aligned (coplanar sensors), separated by known distance, same focal length

"Disparity" is the distance between object's projected position in the two images: x - x'

Recall: correspondence problem

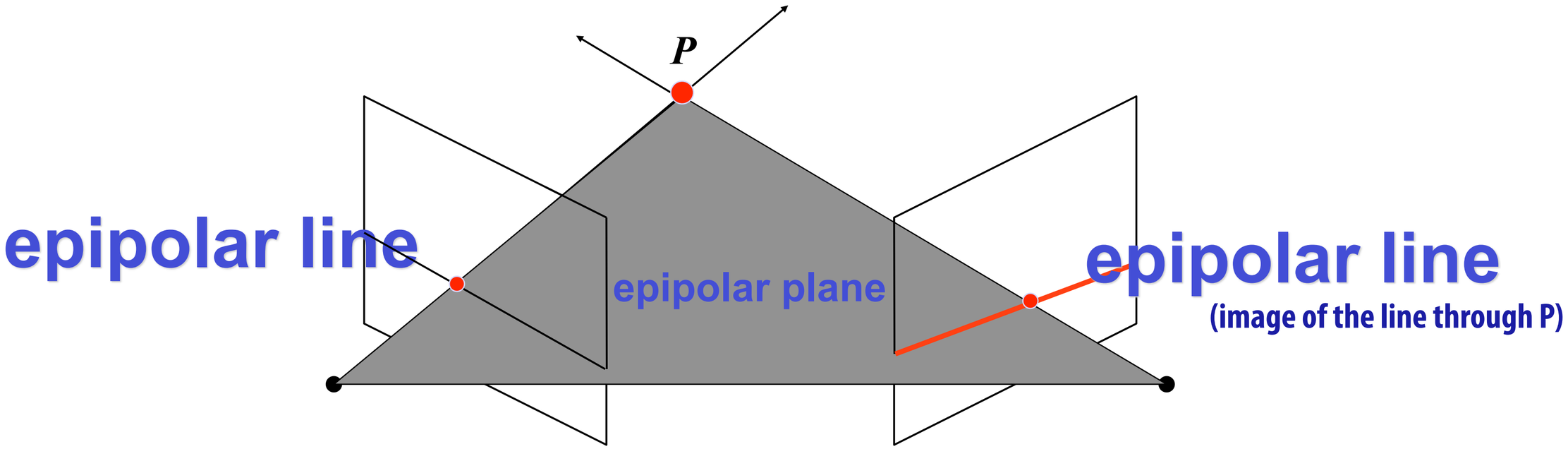
How to determine which pairs of pixels in image 1 and image 2 correspond to the same scene point?



Epipolar constraint

Goal: determine pixel correspondence from pixel values

- Corresponding pixels = pairs of pixels that contain the same scene point

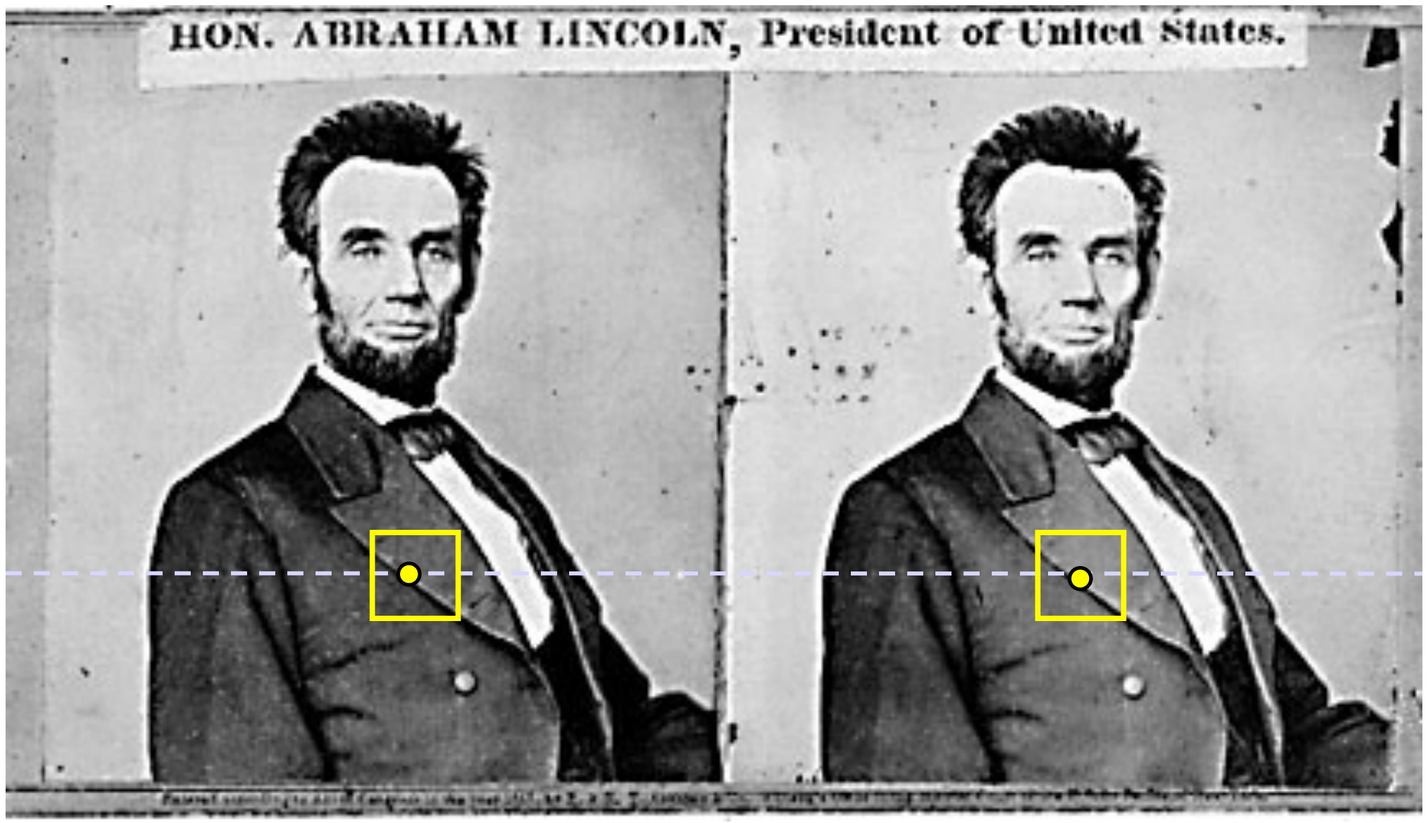


Epipolar Constraint

- Reduces correspondence problem to 1D search along conjugate epipolar lines
- Point in left image will lie on line in right image (epipolar line)

Slide credit: S. Narasimhan

Solving correspondence (basic algorithm)



Slide credit: S. Narasimhan

For each epipolar line

For each pixel in the left image

Compare with every pixel on same epipolar line in right image

Pick pixel with minimum match cost

What are assumptions?

Basic improvements: match windows, adaptive size match windows…

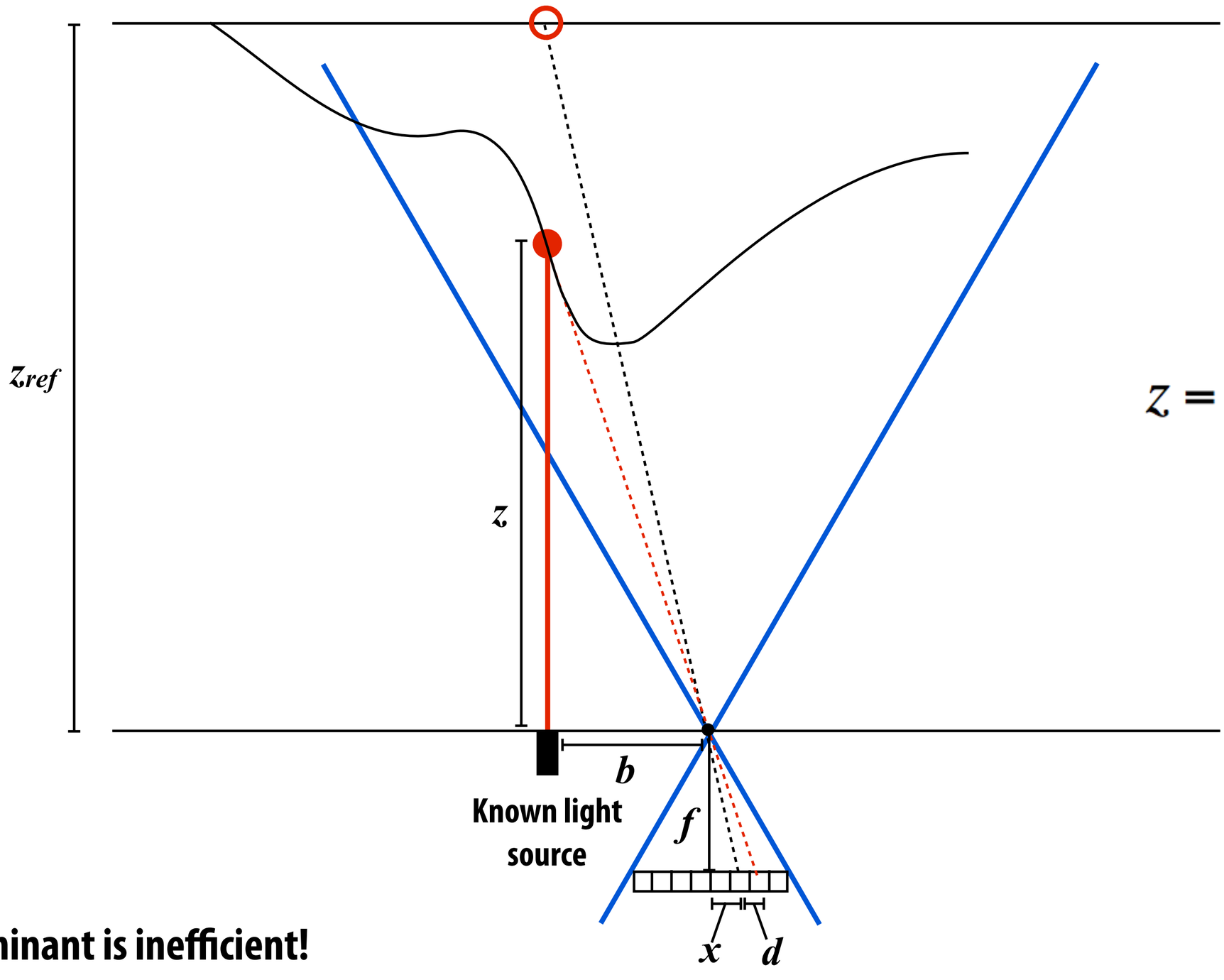
- **This should sound familiar given our discussion of image processing algorithms…**
- **Correlation, sum-of-squared difference (SSD), etc.**

Solving correspondence: robustness challenges

- Scene with no texture (many parts of the scene look the same)
- Non-lambertian surfaces (surface appearance is dependent upon view)
- Pixel pairs may not be present (point on surface is occluded from one view)

Active sensor: emit structured light

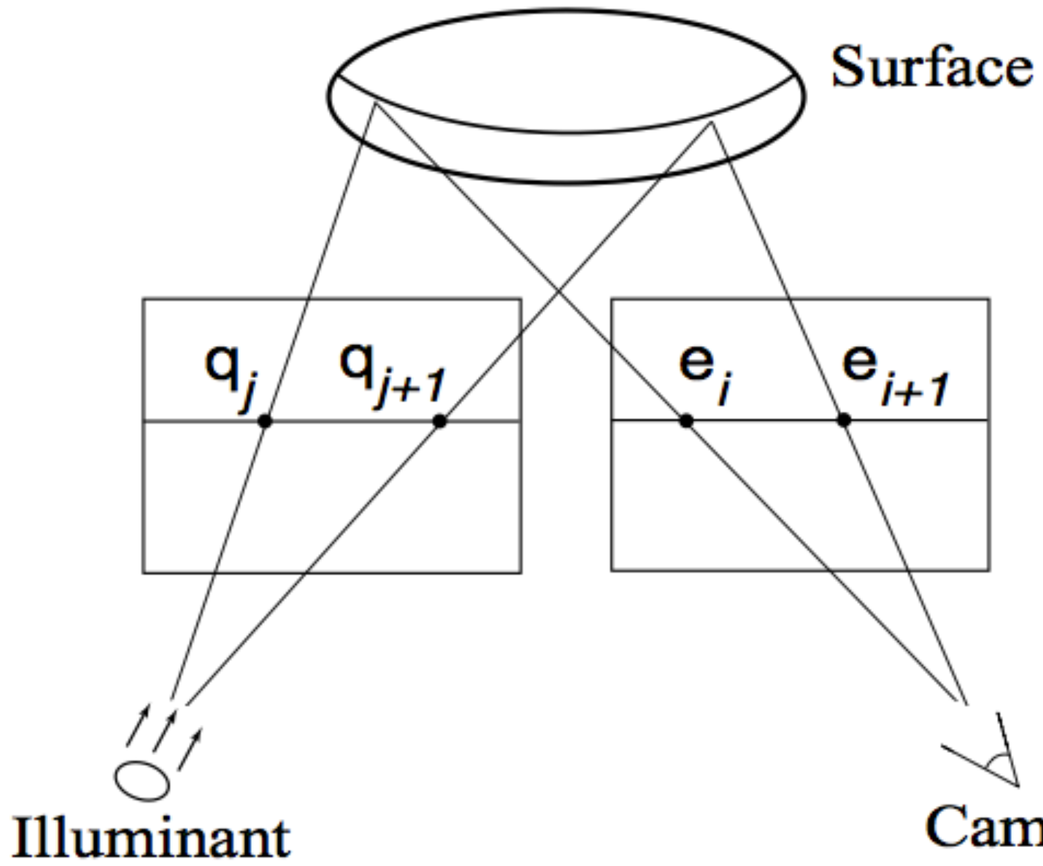
System: one light source emitting known beam + one camera measuring scene appearance
If the scene is at reference plane, image that will be recorded by camera is known
(correspondence between pixel in recorded image and scene point is known)

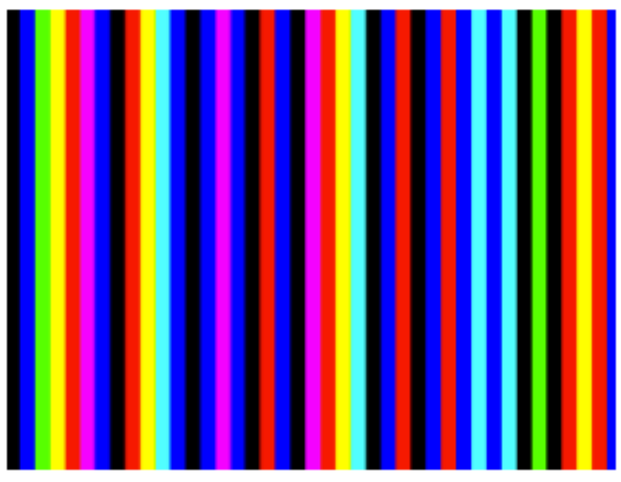


Single spot illuminant is inefficient!
(must "scan" scene with spot to get depth, so high latency to retrieve a single depth image)

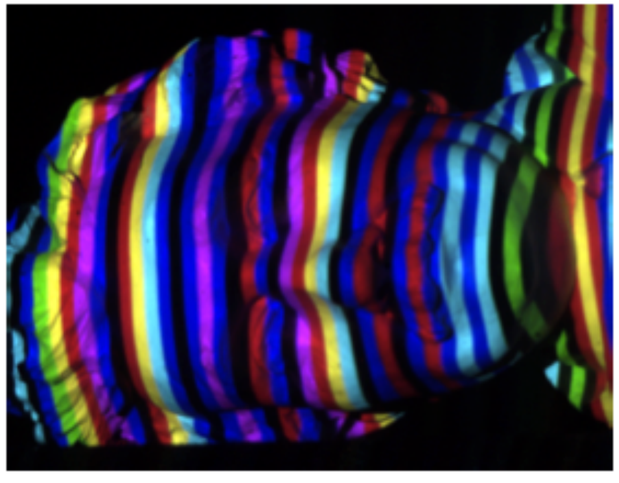
Active sensor: emit structured light

Simplify correspondence problem by encoding spatial position in illuminant





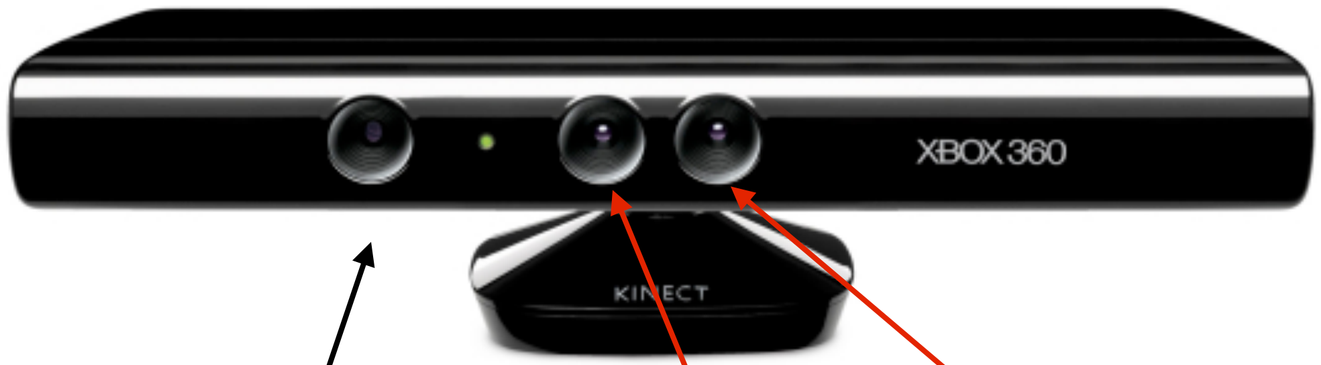
Projected light pattern



Camera image

Image: Zhang et al.

Microsoft XBox 360 Kinect



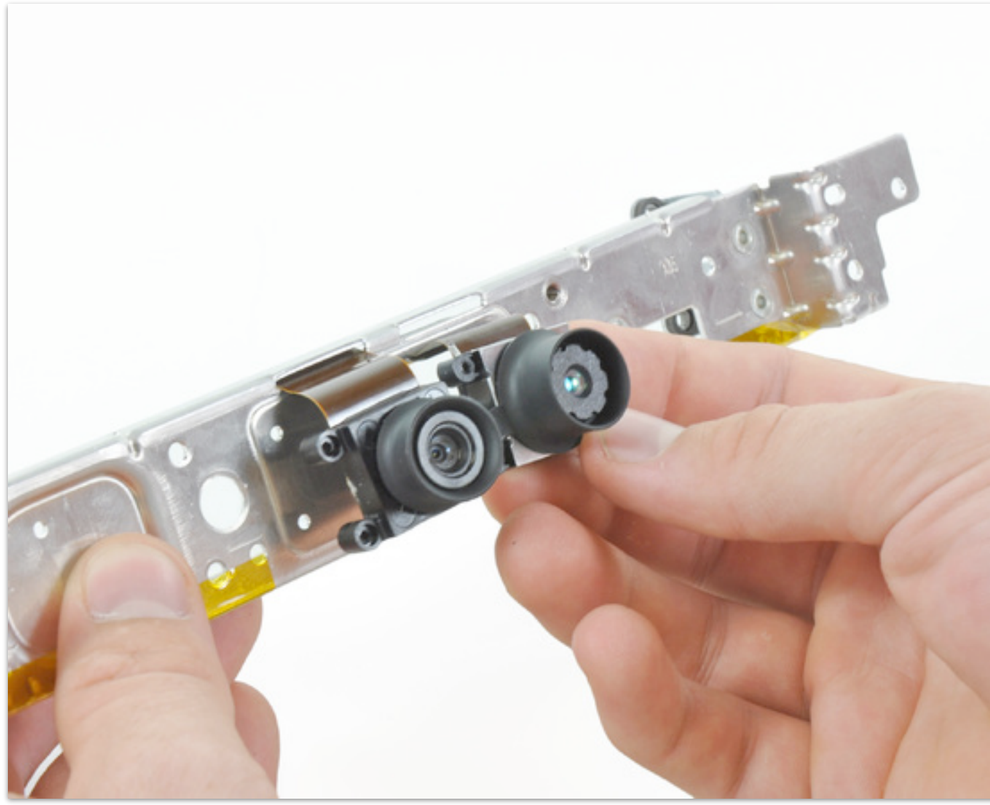


Image credit: iFixIt

Illuminant
(Infrared Laser + diffuser)

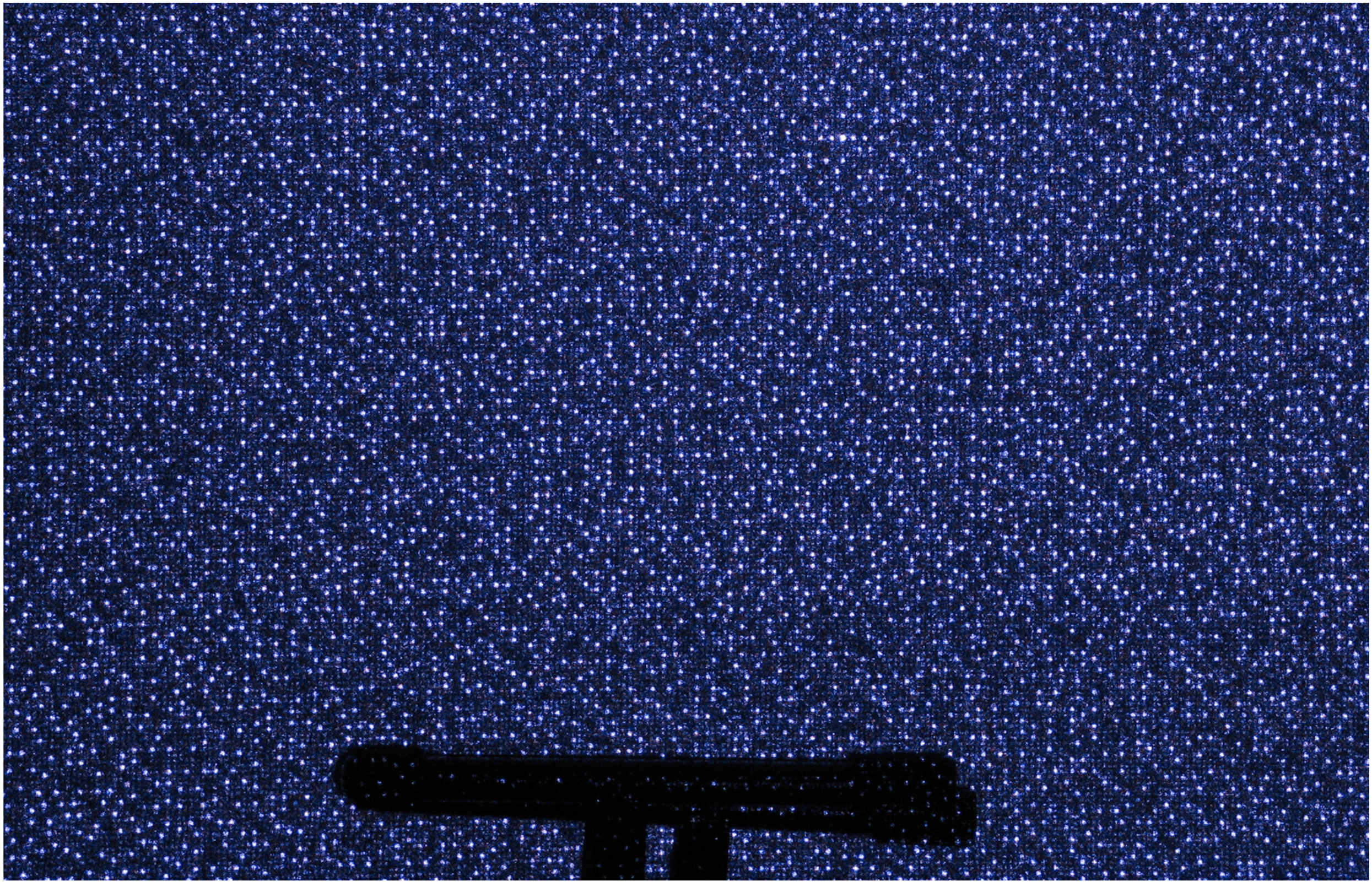
RGB CMOS Sensor
640x480 (w/ Bayer mosaic)

Monochrome Infrared
CMOS Sensor
(Aptina MT9M001)
1280x1024 **

Similar approach used in Intel's RealSense cameras

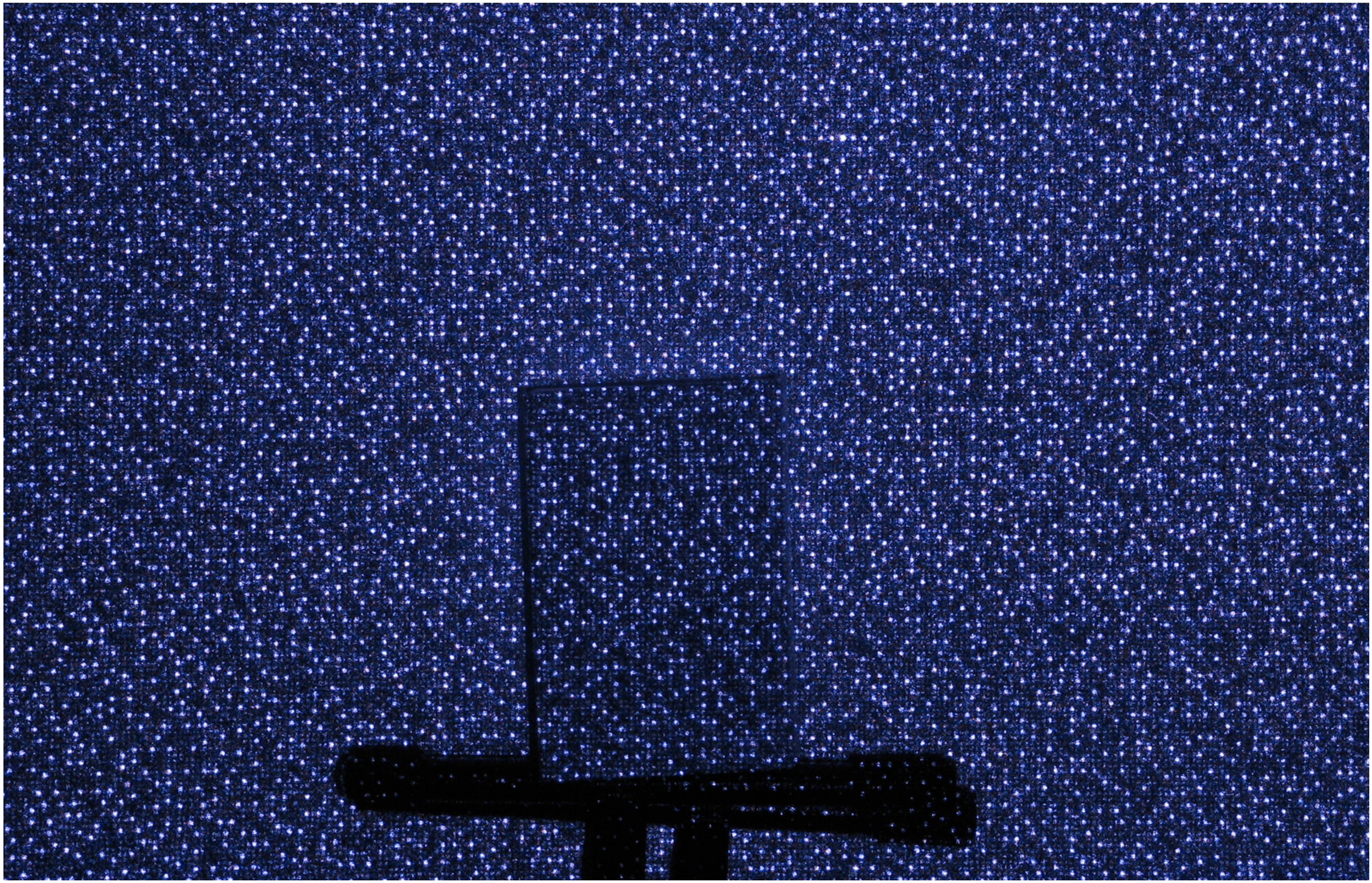
**** Kinect returns 640x480 disparity image, suspect sensor is configured for 2x2 pixel binning down to 640x512, then crop**

Infrared image of Kinect illuminant output



Credit: www.futurepicture.org

Infrared image of Kinect illuminant output



Credit: www.futurepicture.org

Computing disparity for entire scene

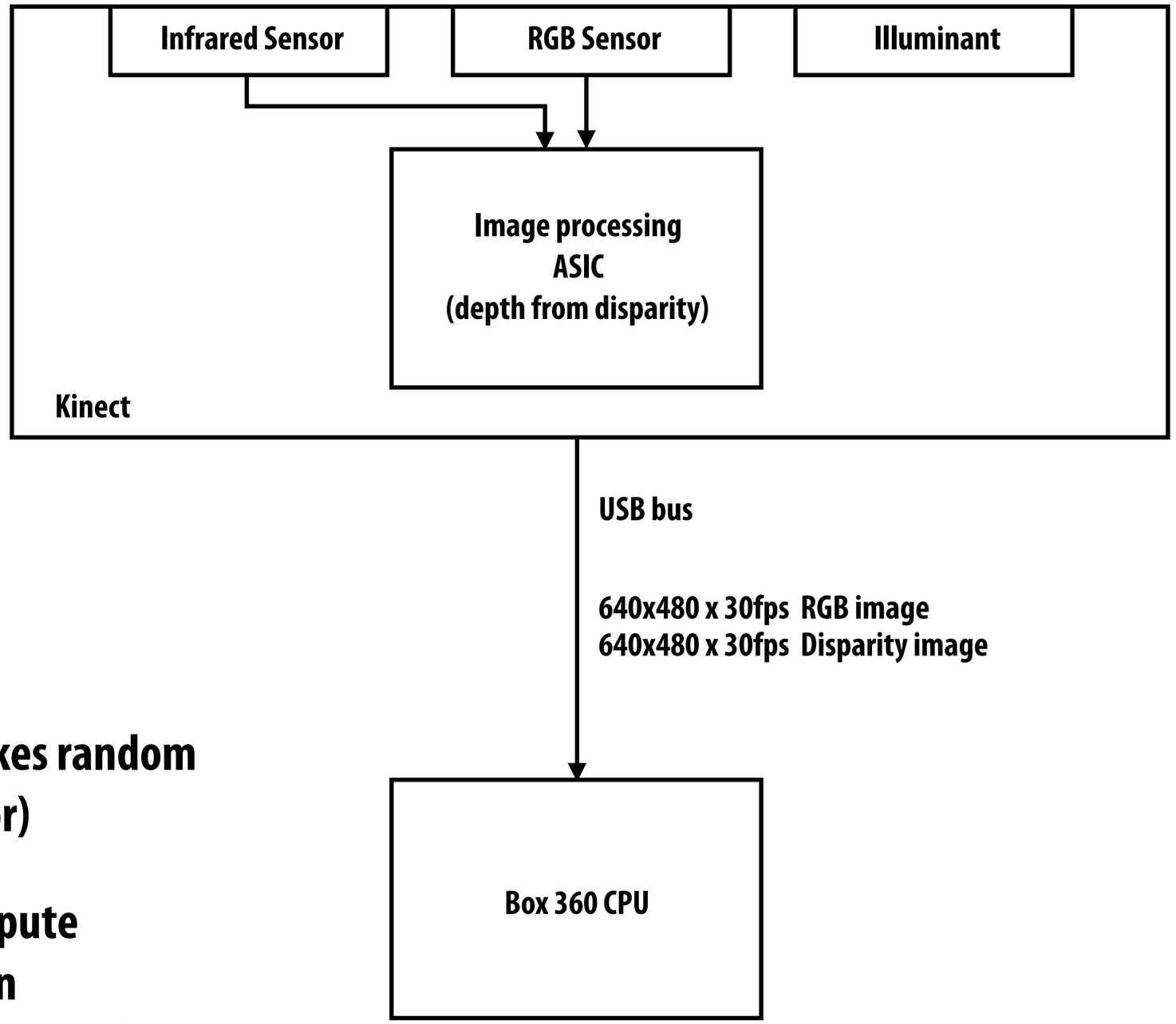
Use region-growing algorithm for compute efficiency *
(Assumption: spatial locality implies depth locality)

1. **Choose output pixels in image, classify as UNKNOWN or SHADOW (based on whether speckle is found)**
2. **While significantly large percentage of output pixels are UNKNOWN**
   - **Choose an UNKNOWN pixel.**
   - **Correlate surrounding MxN pixel window with reference image to compute disparity $D = (dx,dy)$ (note: search window is a horizontal swath of image, plus some vertical slack)**
   - **If sufficiently good correlation is found:**
   - **Mark pixel as a region anchor (its depth is known)**
   - **Attempt to grow region around the anchor pixel:**
     - **Place region anchor in FIFO, mark as ACTIVE**
     - **While FIFO not empty**
     - **Extract pixel P from FIFO (known disparity for P is D)**
     - **Attempt to establish correlations for UNKOWN neighboring pixels P_n of P (left,right,top,bottom neighbors) by searching region given by $P_n + D + (+/-1,+/1)$**
     - **If correlation is found, mark P_n as ACTIVE, set parent to P, add to FIFO**
     - **Else, mark P_n as EDGE, set depth to depth of P.**

*** Source: PrimeSense Patent WO 2007/043036 A1**

Kinect block diagram

Disparity calculations performed by PrimeSense ASIC in Kinect, not by XBox 360 CPU



Cheap sensors: ~ 1 MPixel

Cheap illuminant: laser + diffuser makes random dot pattern (not a traditional projector)

Custom image-processing ASIC to compute disparity image (scale-invariant region correlation involves non-trivial compute cost)

Back to real-time 3D reconstruction

Real-time 3D reconstruction from depth: unique assumptions and challenges

- New assumptions:
  - Now have dense depth estimate at each frame
    - But depth is noisy and may exhibit "holes"
  - Frame-to-frame correspondence made easier since camera undergoes limited motion between frames

- New challenges:
  - Seek dense 3D model
  - Real-time performance (important for providing user accurate feedback during scanning)

Depth image from original Kinect sensor

Image credit: Lejeune et al. 2011

KinectFusion: 3D reconstruction from only depth information

Input: per-frame depth image from frame *k* of capture: $D_k(\mathbf{u})$

(may clean up raw depth from sensor using bilateral filter)

Given depth map and camera calibration information, compute per-frame set of observed surface 3D points (camera-relative coordinates):

$$\mathbf{V}_k(\mathbf{u}) = D_k(\mathbf{K}^{-1}\dot{\mathbf{u}})$$

Given camera-to-world transformation $\mathbf{T}_k$, can compute position of observed points in global coordinate frame
(below: g = "global coordinate frame")

$$\mathbf{V}_k^g(\mathbf{u}) = \mathbf{T}_k\dot{\mathbf{V}}_k(\mathbf{u})$$

How do we estimate camera poses? $\mathbf{T}_k$
How do we fuse partial surface observations into a single consistent 3D model?

$\mathbf{K}$ **= camera intrinsics matrix (focal length, aspect ratio): converts camera-space dir to normalized sensor plane coordinate (pixel)**

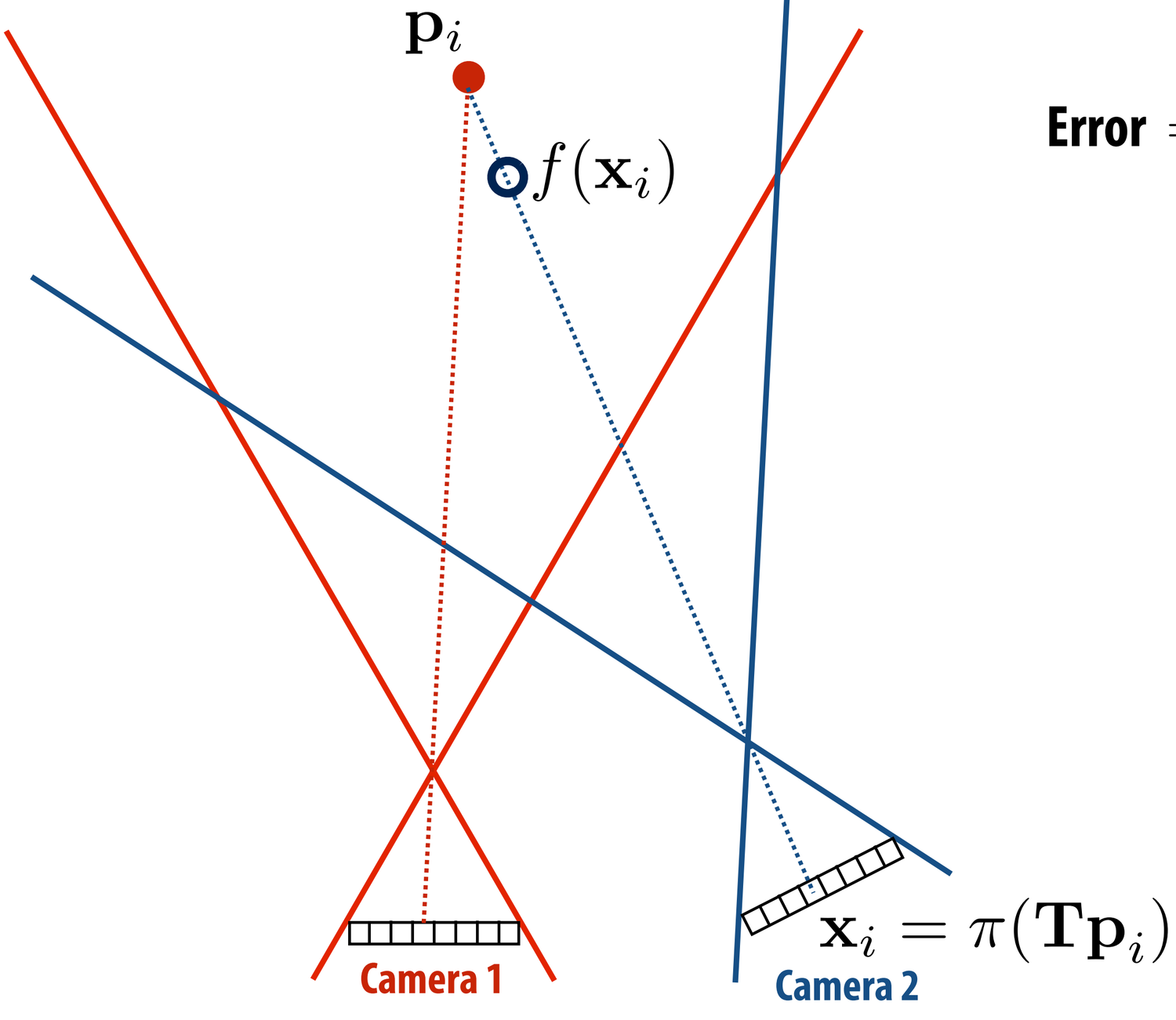
$$\dot{\mathbf{u}} = [\mathbf{u}^T \quad 1]^T$$

$$\dot{\mathbf{V}}_\mathbf{k} = [\mathbf{V}_k^T \quad 1]^T$$

Note: dot notation indicates homogeneous form. [from Newcombe 11]

Fast projective data association (keypoint-less correspondence)

Assume $\mathbf{T}$ moves points from camera 1's reference frame to camera 2's reference frame



Error $= \|f(\mathbf{x}_i) - \mathbf{T}\mathbf{p}_i\|$

$f(\mathbf{x})$ **is actual measurement made by camera 2 at** $\mathbf{x}$

Estimating camera pose ($\mathbf{T}_k$) for new frame using iterated closest point (ICP)

Establish correspondence between depth measurements from frame k with depth estimates from frame k-1: ***

Use each pixel $\mathbf{u}$ in frame *k* with a valid depth estimate, use projective data association to predict corresponding pixel $\hat{\mathbf{u}}$ in frame *k-1*

$$\mathbf{V}_k(\mathbf{u}) = D_k(\mathbf{u})\mathbf{K}^{-1}\dot{\mathbf{u}}$$ ← Measured camera space position of surface in frame k

$$\hat{\mathbf{u}} = \pi\left(\mathbf{T}_{k-1}^{-1}\mathbf{T}_k^z\dot{\mathbf{V}}_k(\mathbf{u})\right)$$

- $\mathbf{T}_k^z\dot{\mathbf{V}}_k(\mathbf{u})$: World-space position of surface as measured by frame k (given current estimate of frame k camera pose)
- $\mathbf{T}_{k-1}^{-1}\mathbf{T}_k^z\dot{\mathbf{V}}_k(\mathbf{u})$: Position of surface relative to camera at frame k-1
- $\hat{\mathbf{u}} = \pi(\ldots)$: Predicted corresponding pixel in frame k-1

$$\hat{\mathbf{V}}_{k-1}^g(\hat{\mathbf{u}}) = \mathbf{T}_{k-1}\hat{\mathbf{V}}_k(\hat{\mathbf{u}})$$ ← World space position of surface(as measured by frame k-1) at predicted corresponding pixel

Seed frame k pose estimate: $\mathbf{T}_k^0 = \mathbf{T}_{k-1}$

***** In practice, use frame-to-model (not frame-to-frame) tracking: establish correspondence with view of volumetric TSDF model from camera position at k-1. See future slides.**

Estimating camera pose ($\mathbf{T}_k$) for new frame using iterated closest point (ICP)

Use "good" correspondences to retain pose estimate:

Depth estimate $D_k(\mathbf{u})$ for pixel $\mathbf{u}$ must be valid

Reasonably good correspondence in position: $\left\|\mathbf{T}_k^z \dot{\mathbf{V}}_k(\mathbf{u}) - \hat{\mathbf{V}}_{k-1}^g(\hat{\mathbf{u}})\right\| \leq \epsilon_d$
(large discrepancy might indicate dis-occlusion)

Reasonably good correspondence in normal (not shown)

Pose error is computed using point-plane metric: (measured surface point at frame *k* should be in surface plane of corresponding point from *k*-1)

$$E(\mathbf{T}_k^z) = \sum_{\mathbf{u} \in \text{valid}} \left[\left(\mathbf{T}_k^z \dot{\mathbf{V}}_k(\mathbf{u}) - \hat{\mathbf{V}}_{k-1}^g(\hat{\mathbf{u}}) \right) \cdot \hat{\mathbf{N}}_{k-1}^g(\hat{\mathbf{u}}) \right]^2$$

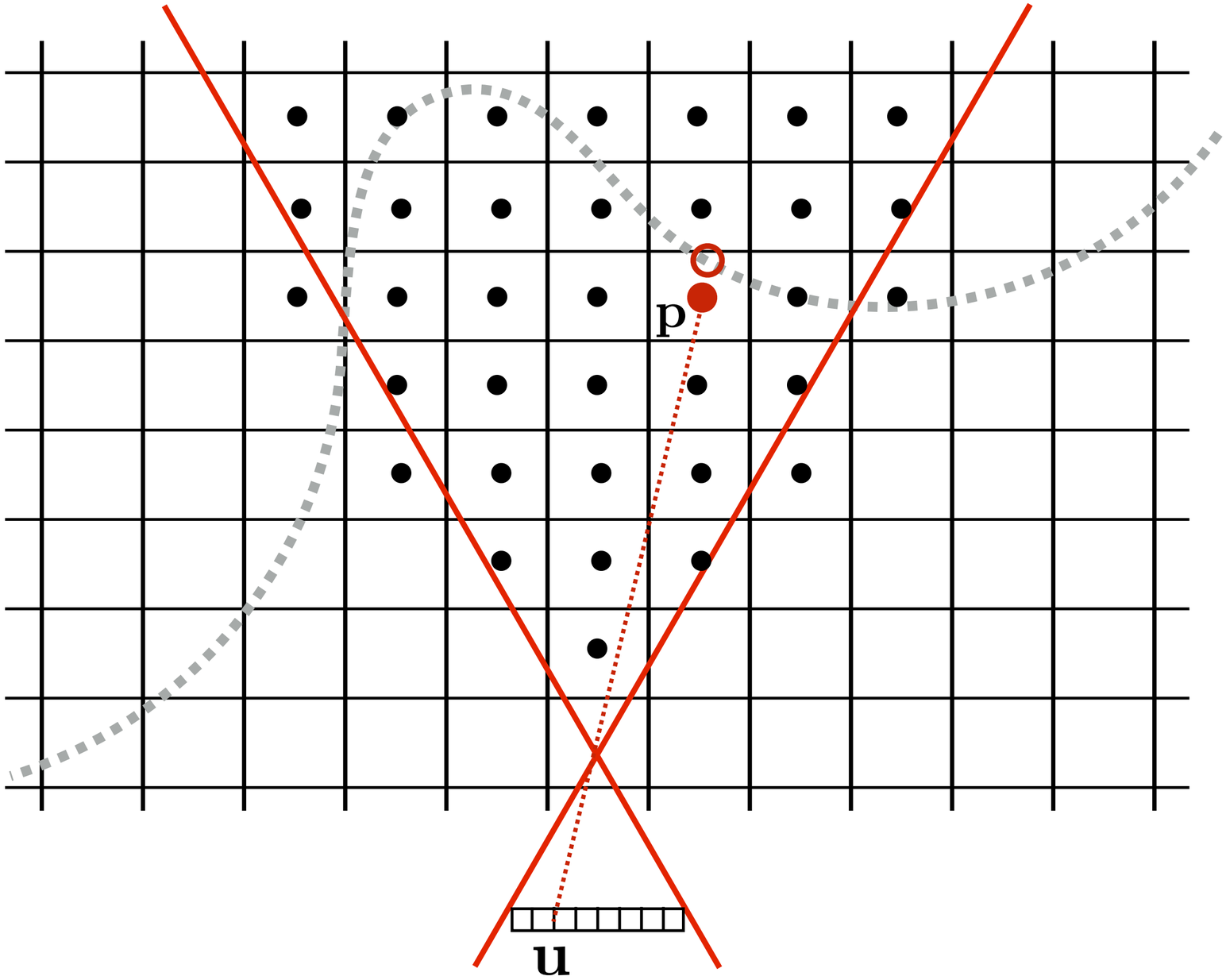
Note: parallel over all pixels

Non-linear optimization to iteratively refine $\mathbf{T}_k$

Computing the 3D surface model

Given camera poses (T_k) and surface vertex estimates ($\mathrm{V}_k^g(\mathbf{u})$) estimate 3D surface position

Store scene geometry in volumetric form as truncated signed distance function (TSDF)



3D voxel grid point *p* is distance $d_{\mathbf{p}}$ from camera, projects to pixel *u*

Camera measures surface at distance $D_k(\mathbf{u})$

Distance is: $d = D_k(\mathbf{u}) - d_{\mathbf{p}}$

Store $\min\left(1, \frac{d}{\mu}\right)$ **grid cell if:**

$$d \geq -\mu$$

Fully data-parallel across voxels

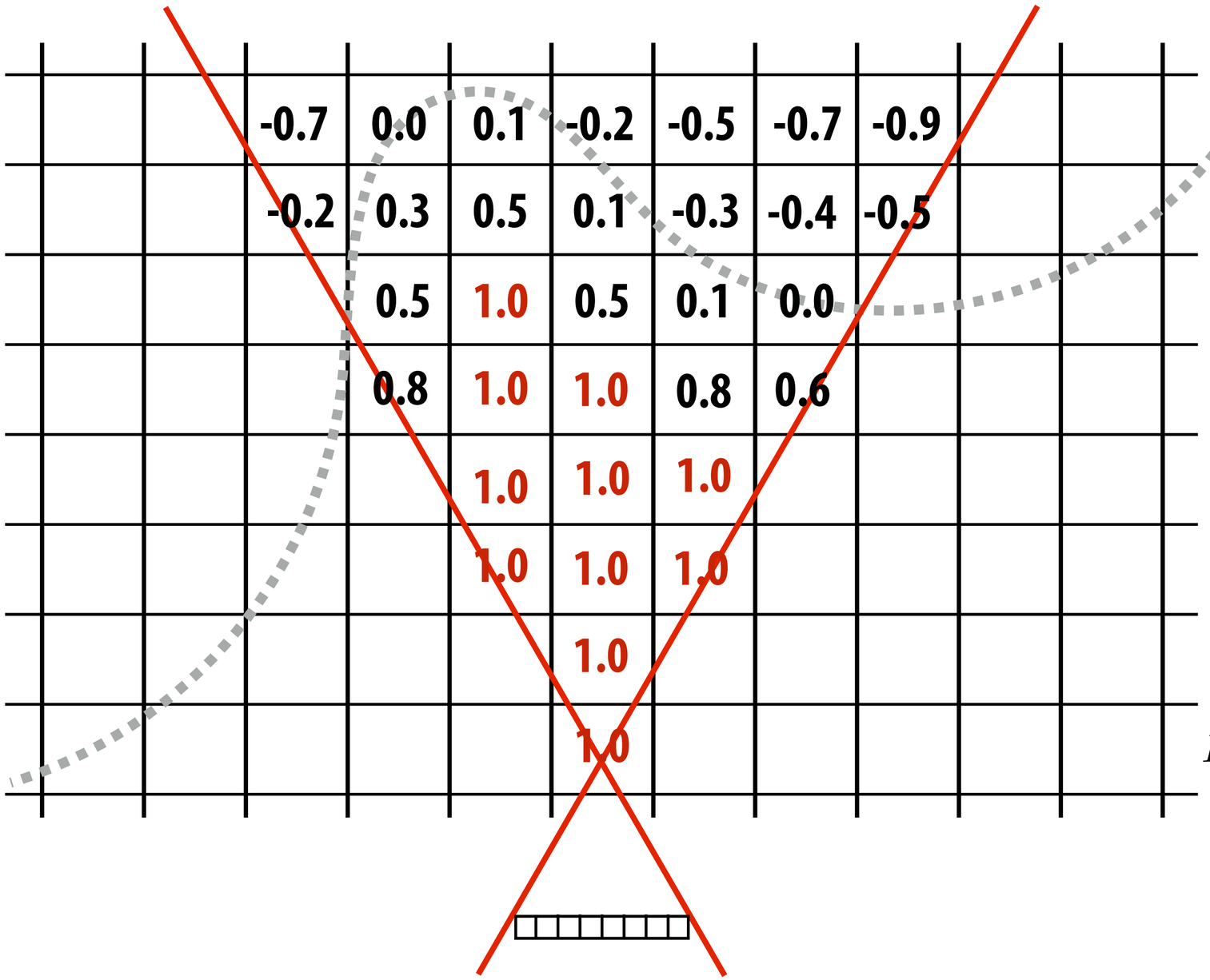
Computing the 3D surface model

Store scene geometry in volumetric form as truncated signed distance function (TSDF)

Each voxel stores:

1. **truncated signed distance:** $F_{D_k}(\mathbf{p})$
2. **weight:** $W_{D_k}(\mathbf{p}) \propto \cos(\theta)/D_k(\mathbf{u})$

Weight closer measurements, and measurements that are not at glancing angles more heavily



F_{D_k} = **contribution to TSDF due to depth map** D_k

Convenient incremental TSDF update when new frames arrive:

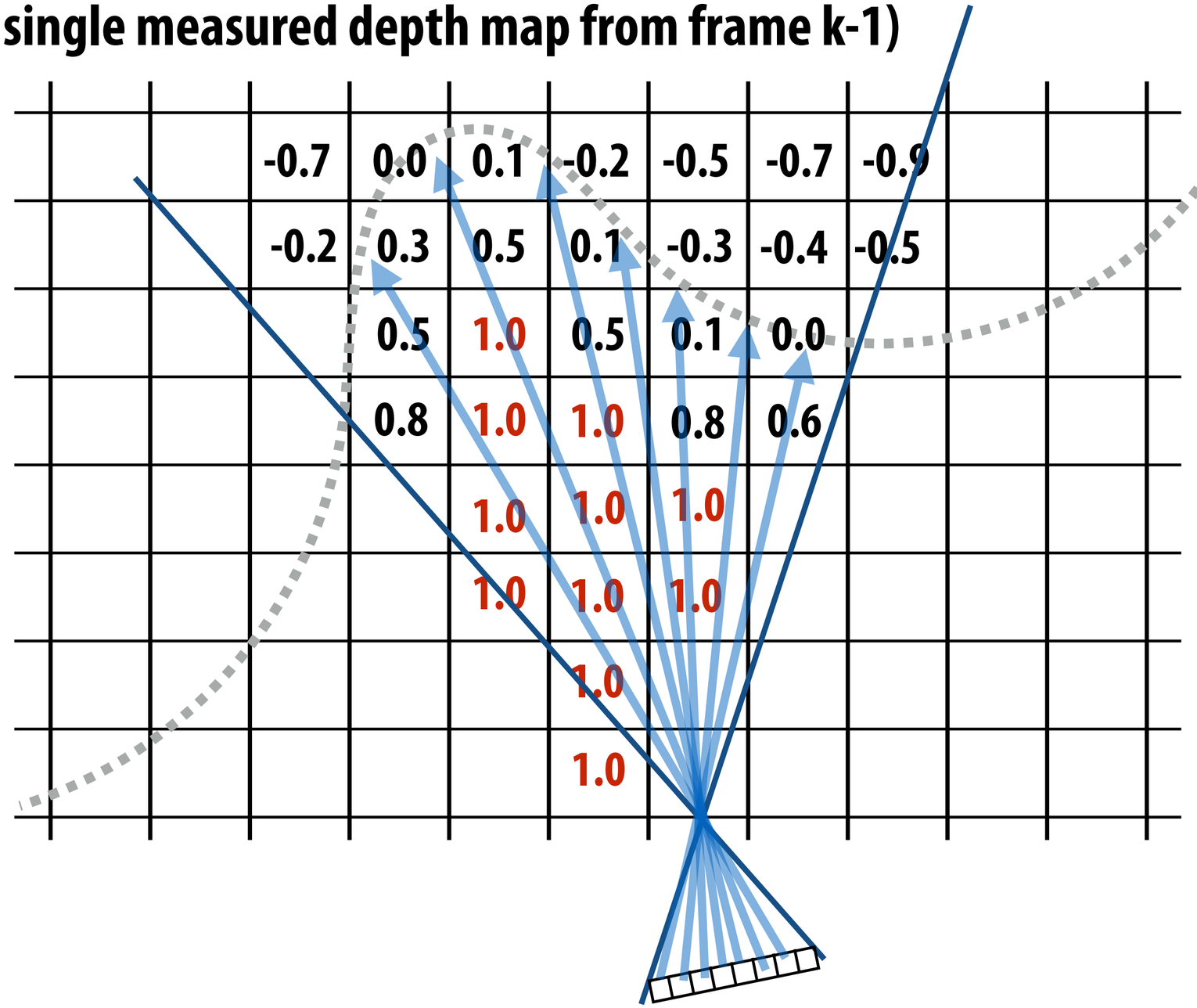
$$F_k(\mathbf{p}) = \frac{W_{k-1}(\mathbf{P})F_k(\mathbf{p}) + W_{D_k}(\mathbf{p})F_{D_k}(\mathbf{p})}{W_{k-1}(\mathbf{p}) + W_{D_k}(\mathbf{p})}$$

$$W_k(\mathbf{p}) = W_{k-1}(\mathbf{p}) + W_{D_k}(\mathbf{p})$$

Computing $D_k(u)$ from the TSDF model (given T^k)

Raycast the TSDF! For each pixel, compute world-coordinate camera ray directions, then march from starting voxel until encountering zero crossing in TSDF

During pose estimation: raycast the TSDF to obtain $\hat{\mathbf{V}}^g_{k-1}(\hat{\mathbf{u}})$ when estimating pose $\mathbf{T}_k$ ("frame-to-model tracking" intuition: TSDF model is more accurate/complete than the single measured depth map from frame k-1)



KinectFusion summary

- Simple data-parallel operations enable efficient real-time GPU implementation
  - Energy term computed in parallel for all pixels (correspondence, error estimation)
  - Each frame updates TSDF independently in parallel over all voxels

- Weighted volumetric surface representation
  - Encodes "uncertainty" in weights
  - TSDF incrementally refines as more frames arrive
    - More confident measurements override incorrect early values
    - Fills "holes" in surface estimate as new views are added
- Not memory efficient (dense 3D voxel representation)
  - KinectFusion (2011) limited to "desk-scale" reconstructions (256^3)

Improving memory efficiency

- One option: adaptive data structures (e.g., octtree)

- Simple and GPU performant solution:
  - Hash table storing non-empty 8^3 voxel-blocks [Nießner 2013]
    $H(x, y, z) = x \cdot p_1 \oplus y \cdot p_2 \oplus z \cdot p_3$

- Upon new frame arrival:
  - Pre-allocate necessary voxelblocks: raycast voxel grid from camera to frame's depth samples to find intersections with voxelblocks
  - Determine voxels to update: for each voxel (in parallel), compute whether voxel is in view frustum (then compact list of touched voxels using standard prefix scan techniques)
  - Update voxels: for each voxel in list (in parallel), perform update similar to dense voxel method

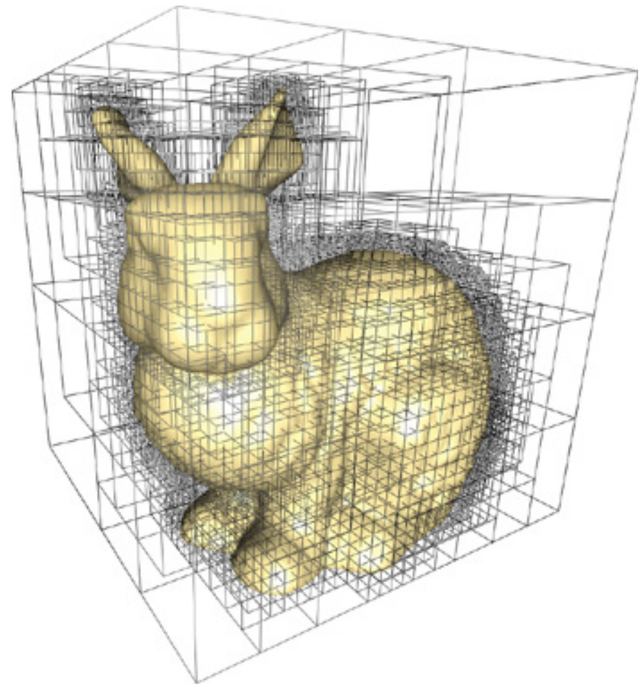
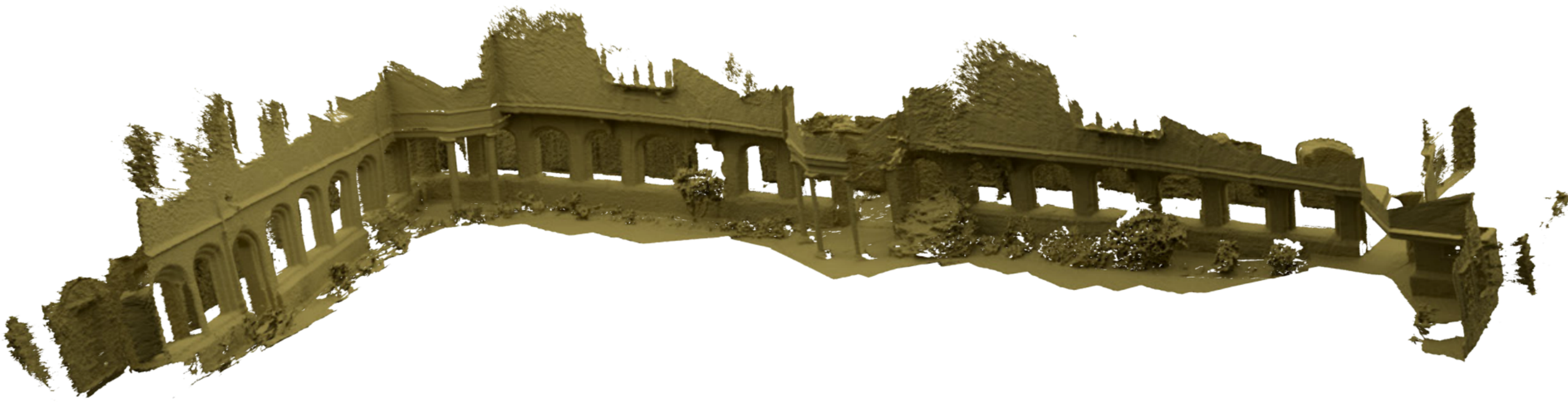


Image credit: Lefebvre et al. GPU Gems 2

Hashing-based sparse voxel representation

On a Titan X GPU: sparsity enables storage of 8mm^2 voxels for near-room sized scenes



[Nießner 2013]

BundleFusion

[Dai 2016]

- Combine incremental depth-based volumetric methods (like we just discussed)
  - Produce dense 3D geometry
  - Performant
  - Quality issues: prone to drift (reliance on temporal tracking)

- With RGB keypoint-based sparse methods (like the offline reconstruction methods discussed in prior lecture)
  - Sparse 3D keypoints
  - Offline: expensive image matching + global optimization
  - Global model consistency via bundle-adjustment (global optimization)

- Goals: high-quality reconstruction, interactive feedback (even for large scans)

BundleFusion: three main ideas

[Dai 2016]

- Sparse-to-dense: use sparse SIFT features to coarsely estimate camera poses, then use dense photometric/geometric consistency for fine-scale alignment

- Use full history of frames (not just recent history) to estimate pose
  - Using full history is made tenable by using two-level hierarchy: align short sequences of frames locality (temporal tracking), then use a representative keyframe from the sequence in global alignment

- "Reintegration" during volumetric fusion: ability to take contribution of a frame out of fused TSDF estimate whenever it's reoptimized pose changes notably (retain high quality TSDF)

Fast sparse+dense correspondence

Given set of frames with per-pixel color (C_i) and depth (D_i), find corresponding features

#Features	Time Detect (ms)	Time Match (ms)
150	3.8	0.04
250	4.2	0.07
1000	5.8	0.42

Sparse

Compute SIFT keypoints for all frames

For each image pair (i,j), use a brute force matching solution to find potential correspondences (no acceleration structure)

Perform geometric verification of correspondences in 3D to estimate rigid camera transform between pair: $\mathbf{T}_{ij} = \mathbf{T}_j^{-1}\mathbf{T}_i$

Discard SIFT keypoint correspondences that are not inliers

Dense

Attempt dense verification using 80x60 "tiny images": C_i^{low} **,** D_i^{low}

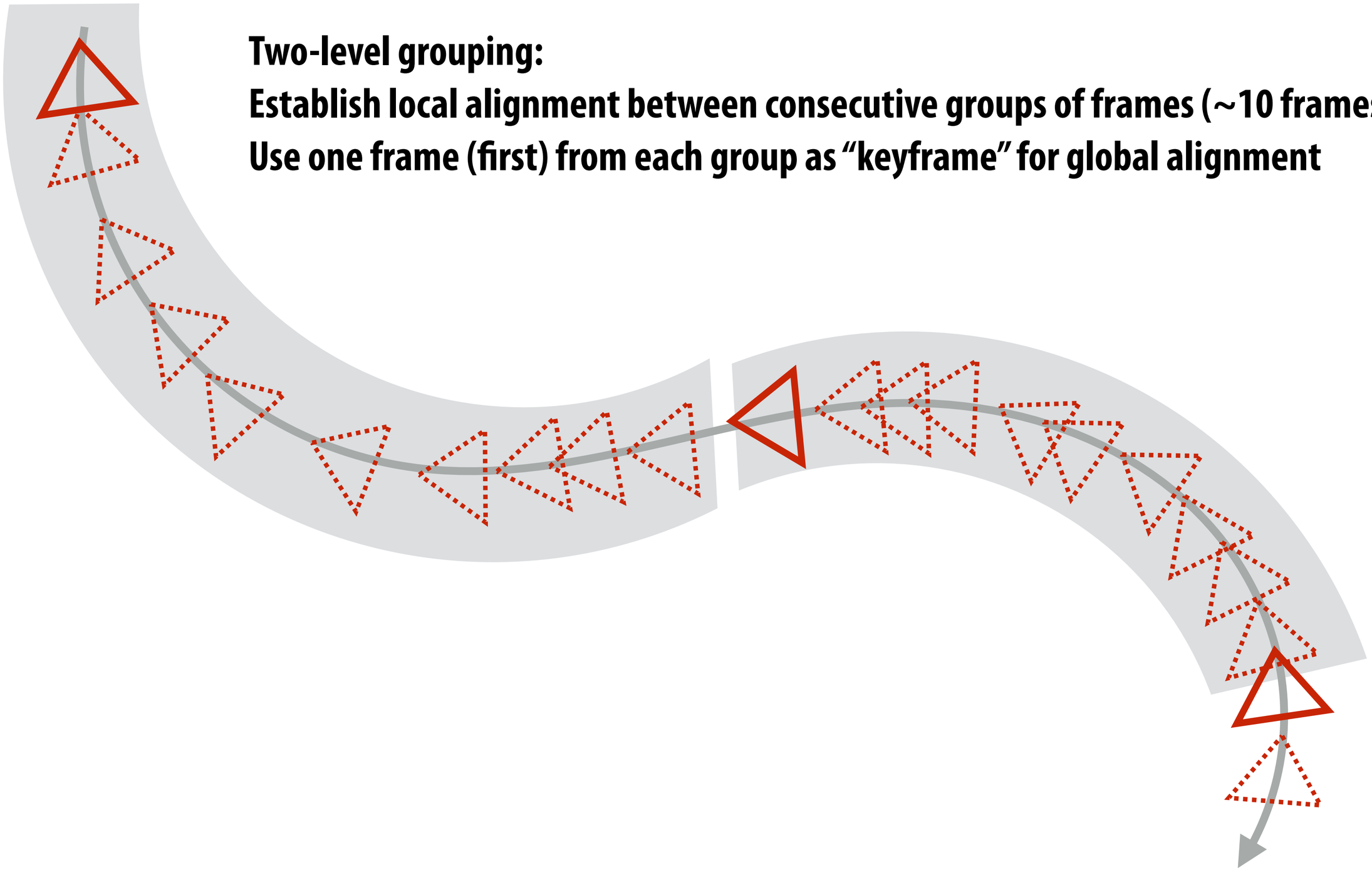
Let $P_i^{\text{low}}(x,y)$ **be measured surface position (relative to camera i) derived from** D_i^{low}

$$E(i,j) = \sum_{x,y} \| \underbrace{\mathbf{T}_{ij} P_i^{\text{low}}(x,y)}_{\text{surface position in camera j's frame (as measured by i)}} - P_j^{\text{low}}(\pi(\underbrace{\mathbf{T}_{ij} P_i^{\text{low}}(x,y)}_{\text{corresponding pixel in image j}})) \|_2$$

Discard match if reproduction error exceeded world-space threshold.

(Similar dense correspondence checks for color and normals)

Hierarchical camera pose estimation



Two-level grouping:
Establish local alignment between consecutive groups of frames (~10 frames)
Use one frame (first) from each group as "keyframe" for global alignment

Sparse + dense energy optimization

Consider S frames in a local frame group (or all S keyframes across entire recording)

Seek camera poses for all frames

$$\chi = [\mathbf{R}_0, \mathbf{t}_0, ..., \mathbf{R}_{S-1}, \mathbf{t}_{S-1}]^T = [x_0, x_1, x_2, ...x_N]^T$$

Set up as non-linear least squares optimization problem

$$E_{\text{align}}(\mathcal{X}) = w_{\text{sparse}} E_{\text{sparse}}(\mathcal{X}) + w_{\text{dense}} E_{\text{dense}}(\mathcal{X})$$

Sparse matching: minimize euclidean distance between all corresponding keypoints.

$$E_{\text{sparse}}(\mathcal{X}) = \sum_{i=1}^{|\mathbf{S}|} \sum_{j=1}^{|\mathbf{S}|} \sum_{(k,l)\in\mathbf{C}(i,j)} \left\| \mathcal{T}_i \mathbf{p}_{i,k} - \mathcal{T}_j \mathbf{p}_{j,l} \right\|^2$$

matching keypoints in frames *i* and *j*

$\mathbf{p}_{i,k}$ **= position of kth key point in ith frame**

Sparse + dense energy optimization

Consider S frames in a local frame group (or all S keyframes across entire recording)

Seek camera poses for all frames

$$\chi = [\mathbf{R}_0, \mathbf{t}_0, ..., \mathbf{R}_{S-1}, \mathbf{t}_{S-1}]^T = [x_0, x_1, x_2, ...x_N]^T$$

Set up as non-linear least squares optimization problem

$$E_{\text{align}}(\mathcal{X}) = w_{\text{sparse}} E_{\text{sparse}}(\mathcal{X}) + w_{\text{dense}} E_{\text{dense}}(\mathcal{X})$$

$$E_{\text{dense}}(\chi) = w_{\text{photo}} E_{\text{photo}}(\chi) + w_{\text{geo}} E_{\text{geo}}(\chi)$$

Dense: photometric alignment (based on matching image gradients (not pixel values)

$$E_{\text{photo}}(\mathcal{X}) = \sum_{(i,j)\in\mathbf{E}} \sum_{k=0}^{|\mathcal{I}_i|} \left\| \mathcal{I}_i(\pi(\mathbf{d}_{i,k})) - \mathcal{I}_j(\pi(\mathcal{T}_j^{-1}\mathcal{T}_i\mathbf{d}_{i,k})) \right\|_2^2$$

tiny image intensity gradients

Dense: geometric alignment (based on point-to-plane metric)

$$E_{\text{geo}}(\chi) = \sum_{(i,j)\in\mathbf{E}} \sum_{k=0}^{|\mathcal{D}_i|} \left[\mathbf{n}_{i,k}^T (\mathbf{d}_{i,k} - \mathcal{T}_i^{-1}\mathcal{T}_j \pi^{-1} \left(\mathcal{D}_j \left(\pi(\mathcal{T}_j^{-1}\mathcal{T}_i \mathbf{d}_{i,k}) \right) \right)) \right]^2$$

Sparse to dense optimization

- **Non-linear least squares problem is linearized and solved using custom iterative solver (conjugate gradient)**

$$E_{\text{align}}(\mathcal{X}) = w_{\text{sparse}} E_{\text{sparse}}(\mathcal{X}) + w_{\text{dense}} E_{\text{dense}}(\mathcal{X})$$

- **Custom: sparse matrix non-zeros computed on-demand (save bandwidth, exploit common subexpressions)**
  - **Fast GPU-based implementation**
  - **See tonights reading…**

- w_{dense} **is increased as the solve proceeds: sparse-to-dense**

Overall flow

- For each group of frames (~10)
  - Extract SIFT features, compute tiny images, compute correspondences (cache tiny images and features of the keyframe)
  - Run optimization described on previous slide to estimate pose
- If group aligns well, add keyframe to global group
  - Run optimizations on all keyframes to globally estimate keyframe poses
  - If keyframe pose changes dramatically, may remove and reinsert frames into voxelized TSDF (TSDF always reflects best globally optimized camera pose estimates)

Inserting a depth frame: (using current best-estimate pose)

$$\mathbf{D}'(\mathbf{v}) = \frac{\mathbf{D}(\mathbf{v})\mathbf{W}(\mathbf{v}) + w_i(\mathbf{v})d_i(\mathbf{v})}{\mathbf{W}(\mathbf{v}) + w_i(\mathbf{v})}, \quad \mathbf{W}'(\mathbf{v}) = \mathbf{W}(\mathbf{v}) + w_i(\mathbf{v}).$$

Removing a depth frame: (using same pose as used to insert)

$$\mathbf{D}'(\mathbf{v}) = \frac{\mathbf{D}(\mathbf{v})\mathbf{W}(\mathbf{v}) - w_i(\mathbf{v})d_i(\mathbf{v})}{\mathbf{W}(\mathbf{v}) - w_i(\mathbf{v})}, \quad \mathbf{W}'(\mathbf{v}) = \mathbf{W}(\mathbf{v}) - w_i(\mathbf{v})$$

Summary

- Modern real-time 3D reconstruction methods mix sparse and dense techniques
- Fast GPU-implementations
  - Brute-force sparse correspondence
  - Data-parallel (pixel-wise) computation of energy terms or voxel-wise TSDF updates / ray marching
- Voxelized 3d model representations support fast incremental update/refinement
  - Memory footprint issues: hashing is GPU-friendly sparse representation du jour
- Global optimization via customized solvers for non-linear least squares problems