CHAPTER 40

Learning to Make Point Clouds

Assume you have a set of images of a scene. You could produce a per-image depth map by just passing each image through a single image depth predictor (Section 36.3). Doing so ignores a great deal of useful information – whenever something appears in more than one image, you can in principle reconstruct it using the methods of Chapters 36.3 if you can be sure about correspondences. But the requirement that you are sure about correspondences forces these methods to produce sparse reconstructions. You could apply the multi-view stereo methods of Chapter 36.3 to a scene graph to get a more dense reconstruction. These methods will produce a dense reconstruction, but require that you have multiple views. Further, they react badly to scenes with moving objects, sometimes called a *4D scene*. Some methods will just produce bad reconstructions. Others will regard moving objects as outliers and ignore them. None is happy to produce a reconstruction of the scene and the moving objects.

Ideally, you want a model that accepts multiple images and produces a reconstruction. A good model would use multiple view information effectively when it was available. It would use single view depth prediction (or something like it) when it had to. It would produce reasonable predictions for scenes with moving objects. Up until very recently, such models were just not available. There is now a deluge of them.

40.1 PRELIMINARIES

40.1.1 Point Maps

A core idea is the point map.

Definition: 40.1 *Point map*

A *point map* for an $M \times N$ image is an $M \times N \times 3$ tensor. Think of the point map as a collection of three vectors, where the three vector at the i, j'th location of the point map encodes the location of the point in 3D that projects to the i, j'th pixel.

The coordinate system in which the point is represented is important, and must be chosen in advance. One natural choice is the coordinate system of the camera that took the image. Assume you have an image obtained in a camera with intrinsic matrix $\mathcal{K}$, and the point map is represented in the camera's coordinate system (focal point at the origin, image plane at $z = f$, and so on). Write d_{ij} for the depth to the i, j'th location in the image. If the point map for the image was

correct, then the i, j'th location would contain

$$\mathcal{K}^{-1} \begin{bmatrix} id_{ij} \\ jd_{ij} \\ d_{ij} \end{bmatrix}.$$

exercises An extremely powerful construction reports the point map for a camera *in a different camera's coordinate system.*

40.1.2 Ray Maps

Another important device is the ray map, which is usually taken together with a depth map.

Definition: 40.2 *Ray map*

A *ray map* for an $M \times N$ image is an $M \times N \times 6$ tensor representing the ray through each pixel location in the camera in the world coordinate system. Interpret the i, j'th element of the ray map as two 3 vectors. The vector represented by the first 3 components gives the origin of the ray (write $\mathbf{t}_{ij}$). The vector represented by the second 3 components gives the direction (write $\mathbf{d}_{ij}$). Use the i, j'th component D_{ij} of the $M \times N$ depth map to interpret the ray map. The point in world coordinates projecting to pixel i, j is given by

$$\mathbf{t}_{ij} + D_{ij}\mathbf{d}_{ij}.$$

There is redundancy in the ray map. For an ideal camera, all the rays should pass through a single point. It is usual to interpret the first three components of the map as the focal point – the origin of the rays – so the first three components of the map should have the same value at each location **exercises**. This redundancy appears to be helpful in learning.

40.1.3 Camera Parameters from a Ray Map

You can recover camera parameters from a ray map. Recovering the focal point is straightforward.

Procedure: 40.1 *Focal point from ray map*

Estimate the camera's focal point in world coordinates by

$$\hat{\mathbf{f}} = \frac{1}{MN} \sum_{ij} \mathbf{t}_{ij}.$$

Camera intrinsics and rotation can be recovered with a homography estimate. Assume that the focal point estimate is correct **exercises**. In world coordinates,

the camera is $\mathcal{K}\left[\mathcal{R}| - \mathcal{R}\hat{\mathbf{f}}\right]$. The i, j'th 3D point projects to (x_{ij}, y_{ij}) in the image. Write the 3D point in world coordinates using homogenous coordinates, apply the camera, and obtain

$$\begin{bmatrix} x_{ij} \\ y_{ij} \\ 1 \end{bmatrix} \equiv \mathcal{K}\left[\mathcal{R}| - \mathcal{R}\hat{\mathbf{f}}\right] \begin{bmatrix} \hat{\mathbf{f}} + D_{ij}\mathbf{d}_{ij} \\ 1 \end{bmatrix}.$$

This simplifies to

$$\begin{bmatrix} x_{ij} \\ y_{ij} \\ 1 \end{bmatrix} \equiv \mathcal{K}\mathcal{R}\mathbf{d}_{ij} \equiv \mathcal{H}\mathbf{d}_{ij}.$$

exercises

Procedure: 40.2 *Camera rotation and intrinsics from ray map*

Using the notation above and

$$\begin{bmatrix} x_{ij} \\ y_{ij} \\ 1 \end{bmatrix} \equiv \mathcal{H}\mathbf{d}_{ij}$$

fit a homography $\mathcal{H}$. Obtain the $\mathbf{d}_{ij}$ from the ray map. Now factor the homography as in Section 36.3 to obtain the rotation matrix and the intrinsics.

40.2 EVALUATING RECONSTRUCTIONS

At a high level, evaluation is easy: you evaluate a reconstruction by comparing what you reconstructed with what you should have reconstructed. The details present important difficulties. You must choose a metric to compare the two. You need to have a ground truth to compare to, and these can be difficult to obtain. Proxy evaluation – where you evaluate something that would have to be right if the reconstruction were right – can help when ground truth is hard to get.

40.2.1 Evaluating Per-Image Depth Maps

Evaluating per-image depth maps is relatively straightforward. You use the metrics for single image depth evaluation (AbsRel and δ) given in Section 19.2.2.

Procedure: 40.3 *Evaluating per-image depth maps*

Compute AbsRel and δ for each predicted depth maps, then average over images. Good practice reports the standard deviation for each as well.

40.2.2 The Chamfer Distance

Comparing two point clouds is tricky. The point clouds should be in the same coordinate system for a meaningful comparison, but this is easily dealt with (use the methods of Chapter 36.3). The complications occur because there is no reason to expect that the point clouds correspond.

Write $\mathbf{x}_i$ for the i'th point in $\mathcal{X}$ – the first set of points. The second set is $\mathcal{Y}$, so $\mathbf{y}_i$. The sets are not ordered, so there is no reason to believe that you can meaningfully compare $\mathbf{x}_i$ with $\mathbf{y}_i$. The sets are sampled from surfaces, so there is no reason to believe that there is *any* $\mathbf{y}_j$ such that $\mathbf{x}_i$ should be the same as $\mathbf{y}_j$. They might be sampled with slightly different regimes, too. For example, imagine comparing a point cloud obtained from a distant view of an object with a point cloud obtained from a nearby view. The distant view will have fewer points that are further apart on the object.

The *chamfer distance* is a way to compare sets of points under these circumstances. Unfortunately, there are two versions in circulation, and you need to be very careful about which one any particular experimenter is using.

Procedure: 40.4 *Chamfer distance, v1*

Compute

$$\mathrm{CHD}(\mathcal{X}, \mathcal{Y}) = \sum_{\mathbf{y} \in \mathcal{Y}} \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmin}} \; d(\mathbf{x}, \mathbf{y})$$

where $d(\mathbf{x}, \mathbf{y})$ is an appropriate distance, usually $\left[(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y}) \right]^{1/2}$.

Procedure: 40.5 *Chamfer distance, v2*

Compute

$$\mathrm{CHD}(\mathcal{X}, \mathcal{Y}) = \sum_{\mathbf{y} \in \mathcal{Y}} \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmin}} \; d(\mathbf{x}, \mathbf{y})^2$$

where $d(\mathbf{x}, \mathbf{y})^2$ is the square of an appropriate distance, usually $d(\mathbf{x}, \mathbf{y})^2 = \left[(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y}) \right]$.

Notice that in either case, the distance is not symmetric. Some authors use a symmetrized version

$$\mathrm{CHD}_{\mathrm{symm}}(\mathcal{X}, \mathcal{Y}) = \mathrm{CHD}(\mathcal{X}, \mathcal{Y}) + \mathrm{CHD}(\mathcal{Y}, \mathcal{X}).$$

exercises

You should be concerned that the chamfer distance is expensive to compute for large point clouds (the nearest neighbors step **exercises**). An approximate

algorithm exists [**?**]. This gives a $(1+\epsilon)$ approximation with a running time of $O(nd \log n/\epsilon^2)$ (d is dimension; n the size of the point sets), but ϵ may need to be small to get a reliable ordering of algorithms **exercises**.

40.2.3 Comparing Translation Estimates

Multiple view reconstruction methods must produce an estimate of the translation from camera to camera in some way. Most methods do this explicitly. For some methods, you might need to do some work to recover the translation. For example, methods that produce ray maps may not explicitly produce a camera translation estimate, but (as Section **??** showed) you can recover a translation estimate from the ray map. Recall that reconstructions are up to a scale, so comparing the length of translation vectors is unwise. You cannot compare the locations of camera centers directly. The RTA (relative translation accuracy) metric compares the unit vectors between camera centers to ground truth.

Procedure: 40.6 *Relative translation accuracy, or RTA*

There are N images. Write $\hat{\mathbf{f}}_i$ for the reconstructed camera center for the i'th camera, and $\mathbf{f}_i$ for the ground truth. Write $\hat{\mathbf{t}}_{ij} = \hat{\mathbf{f}}_j - \hat{\mathbf{f}}_i$. Now compute

$$\text{RTA} = \frac{2}{N(N-1)} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \arccos\left(\frac{\hat{\mathbf{t}}_{ij}^T \mathbf{t}_{ij}}{\| \hat{\mathbf{t}}_{ij} \|_2 \| \mathbf{t}_{ij} \|_2} \right)$$

40.2.4 Relative Rotation Accuracy

The camera rotations in reconstruction and ground truth cannot be compared directly, because the reconstruction may be in a different coordinate system from ground truth. You can compare the rotation from camera i to camera j in the reconstruction and in the ground truth, rather like RTA. Doing so requires a metric on rotation matrices.

Definition: 40.3 *Angular distance between rotation matrices*

Given two rotation matrices $\mathcal{R}_A$ and $\mathcal{R}_B$,

$$\text{Adist}(\mathcal{R}_A, \mathcal{R}_B) = \arccos\left(\frac{\text{trace}(\mathcal{R}_A^T \mathcal{R}_B) - 1}{2} \right)$$

is the angular distance between the matrices. It can be shown that this distance correctly measures the distance between two rotation matrices (it correctly reports the length of a geodesic connecting the two rotations on the group of rotation matrices).

Procedure: 40.7 *Relative Rotation Accuracy, or RRA*

There are N images. Write $\hat{\mathcal{R}}_i$ for the reconstructed camera rotation for the i'th camera, and $\mathcal{R}_i$ for the ground truth camera rotation for the i'th camera. These might be in different coordinate systems. Write $\hat{\mathbf{R}}_{ij} = \hat{\mathcal{R}}_j \hat{\mathcal{R}}_i^T$ for the rotation from camera i's coordinate system to camera j's coordinate system (etc.). Now compute

$$\text{RRA} = \frac{2}{N(N-1)} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \text{Adist}\left(\hat{\mathcal{R}}_{ij}, \mathcal{R}_{ij}\right).$$

40.2.5 Missing Metrics

There are some metrics that follow from this discussion that I am not aware of anyone using. The RTA assumes that the reconstruction and ground truth are in the same coordinate system. It is relatively straightforward to produce a metric that does not make this assumption **exercises**, but I am not aware of anyone using it. It is also relatively straightforward to deal with the scaling issue in producing a metric **exercises**. Finally, because you can recover camera intrinsics from point maps (Section 40.3.2), you could compare recovered intrinsics with known intrinsics.

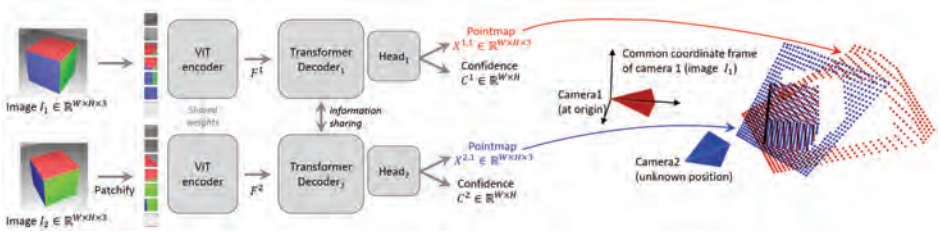


FIGURE 40.1: *DUSt3R passes two images through transformer encoders trained with CroCroV2, then passes the tokens to two dense decoders. Each decoder produces point maps – one 3D point per pixel – and confidence maps. Confidence maps are large when the prediction is more confident. Both point maps (here, the red and blue point clouds) are in the first image's coordinate system. The system is trained with image pairs with known point maps.*
Image credit *Uses Figure 2 of "DUSt3R: Geometric 3D Vision Made Easy," Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii and Jerome Revaud, CVPR, 2024*

40.3 PREDICTING POINT MAPS WITH DUST3R

DUSt3R uses a structure analogous with CroCoV2 (Section 36.2). DUSt3R accepts two images, and encodes them using two copies of a transformer encoder. The trans-

former encoder is pre-trained using the CroCo loss, and so is heavily inclined to produce tokens that support cross-view reconstruction. There are two dense decoders, rather than the one decoder of CroCo stereo or CroCo flow (Section 36.2.3). Each sees encoded tokens from both images (Figure 40.4).

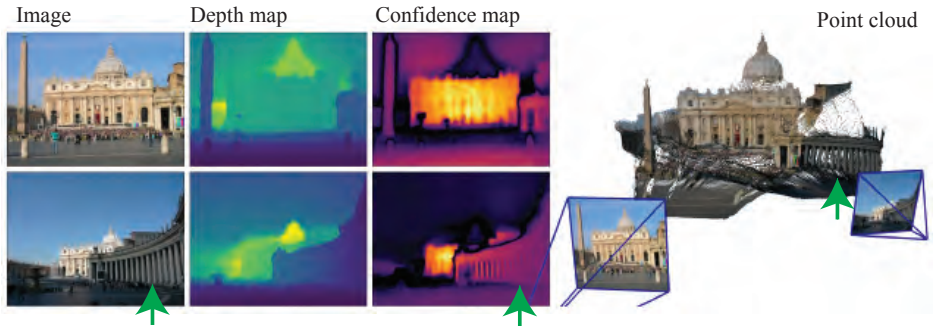


FIGURE 40.2: *DUSt3R reconstructions from a pair of images, showing per image depth maps (bluer is closer), confidence maps (yellower is more confident) and a point cloud with inferred camera configurations. Notice the image pair is quite challenging, with a significant change in view. Some parts of one image are not visible in the other (for example, the facade identified by the* **green** *arrow). These have lower confidence values, and reconstruction is reasonable but not very accurate. The reconstruction must be a single image depth reconstruction.*
Image credit *Uses part of Figure 3 of "DUSt3R: Geometric 3D Vision Made Easy," Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii and Jerome Revaud, CVPR, 2024*

The first image is decoded into a point map representing points in the first image's coordinate system. The crucial, and clever, step here is that the second image's point map is trained to so that the entry in the second image's point map corresponding to a pixel is the point in 3D that projects to the pixel in the second image, *but in the first image's coordinate system* (Figure **??**) **exercises**. Notice that swapping the two input images in DUSt3R will result in significant changes in the point maps **exercises**. Further, DUSt3R is configured to report confidence in the two point maps (much more detail below). This means that it is not forced to make accurate predictions where it can't. For example, if a location appears in one view but not another, any prediction must rely on structural priors and so on. As another example, if you pass it two images that don't share any 3D points you will not get bad outcomes, just very low confidence predictions.

40.3.1 Training DUSt3R

Start with a transformer encoder pre-trained using the CroCo loss. Attach two dense decoders. Now obtain a large number of image pairs showing different views of a scene. You need to know the right point map for both images in the pair. Such data can now be obtained from the internet (see Section **??**), but has to be created in some way. You could use a camera attached to a depth sensor to create this

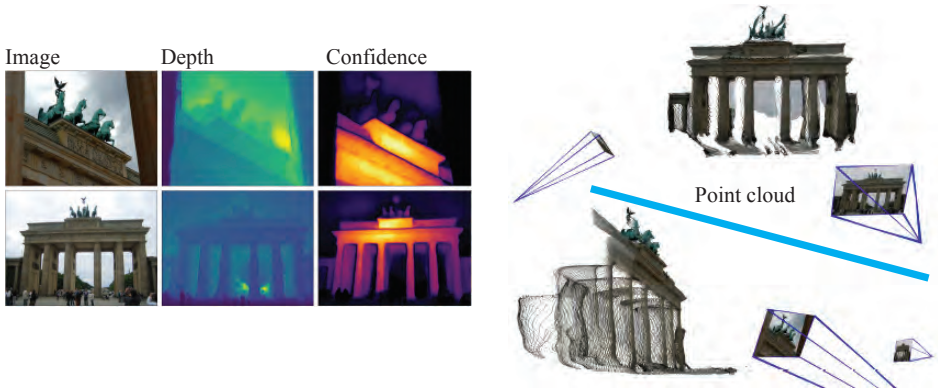


FIGURE 40.3: *Another set of DUSt3R reconstructions from a pair of images, showing per image depth maps (bluer is closer), confidence maps (yellower is more confident). The point cloud is shown viewed from two different views, with inferred camera configurations (the frustrums). Notice the image pair is quite challenging, with a significant change in view. Much of one image – the pillars in the lower image – is not visible in the other. These have lower confidence values, but the reconstruction is rather good – look at the point clouds. This is likely because the image of the pillars appears to allow very good single image depth reconstruction, and because a strong implicit prior on structure helps attach the pillars to the top of the facade.*
Image credit *Constructed from Figure 4 of "DUSt3R: Geometric 3D Vision Made Easy," Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii and Jerome Revaud, CVPR, 2024*

data. You could use renderings of 3D models, obtained either by measurement or by some authoring process. You could use data produced by SFM pipelines (as in Chapter 38). As for CroCo, quite careful curation is required (see Section 36.2.1 and [**?**]).

DUSt3R must reconstruct points from two images where the relative configuration of the cameras is unknown. For the moment, assume the network just spoofs an elementary method, and that the calibration is known. Recall from Section 36.3 that, in this case, you can compute the relative configuration of two cameras up to scale. This suggests (correctly) that DUSt3R's point map predictions will not have the correct scale compared to ground truth. The training loss must take this into account. Training data consists of tuples: a pair of images, and a pair of point maps in the first camera's frame. Scale the pair of ground truth point maps so that the mean distance of points from the origin is one.

To evaluate, scale the pair of predicted point maps so that the mean distance of points from the origin is one. For $v \in \{1, 2\}$, write $X_{ij}^{v,1}$ for the scaled predicted point map corresponding to the v'th image in the first camera's frame at the i, j'th location and use an overline to denote the ground truth (so $\overline{X}_{ij}^{1,1}$ is the scaled ground truth point map, etc.). Write C_{ij}^{v} for the confidence map for the v'th image

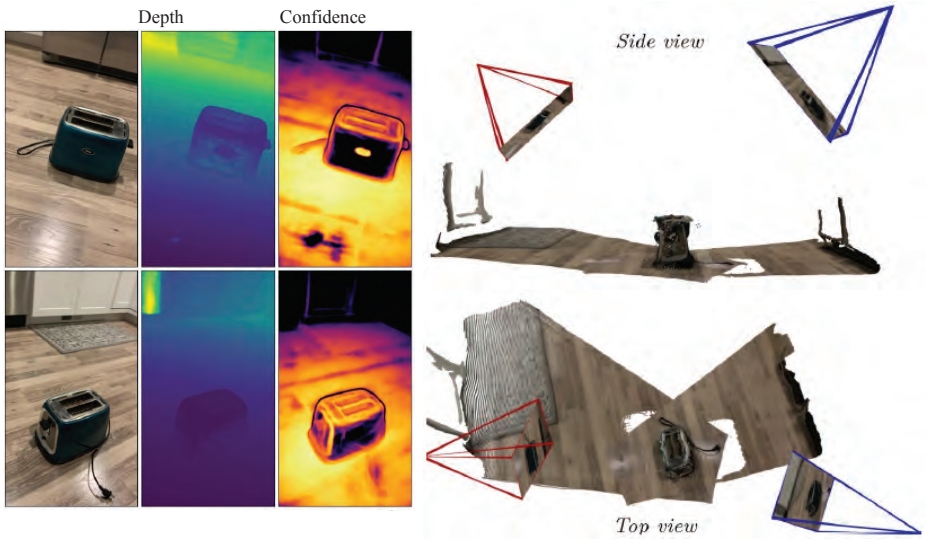


FIGURE 40.4: *Another set of DUSt3R reconstructions from a pair of images, showing per image depth maps (bluer is closer), confidence maps (yellower is more confident). The point cloud is shown viewed from two different views, with inferred camera configurations (the frustrums). Here the change in view is extreme, with the two cameras viewing the toaster from different sides. Nonetheless, the reconstruction is rather good – look at the point clouds. This is likely again the result of very good single image depth reconstruction and a strong implicit prior.*
Image credit *Constructed from parts of Figure 8 of "DUSt3R: Geometric 3D Vision Made Easy," Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii and Jerome Revaud, CVPR, 2024*

at the i, j'th location. The loss is then

$$\sum_{ijv} \left[C_{ij}^v \| X_{ij}^{v,1} - \overline{X}_{ij}^{v,1} \| - \alpha \log C_{ij}^v \right]$$

This loss will be small if either the error is small or the confidence is small, and the $\log C_{ij}^v$ term encourages the network to be confident when the error is small **exercises**. Notice the difference in sign with the expression in Section 36.2.3 **exercises**. The confidence needs to be positive, so predict some Z_{ij}^V and use $C_{ij}^v = 1 + \exp Z_{ij}^v$.

40.3.2 Exploiting DUSt3R

The resulting system is extremely powerful. Write $\mathcal{P}_1(\mathcal{I}_1, \mathcal{I}_2)$ and $\mathcal{P}_2(\mathcal{I}_1, \mathcal{I}_2)$ for the two point maps. Here are some other problems you can solve with the point maps produced by DUSt3R **exercises**.

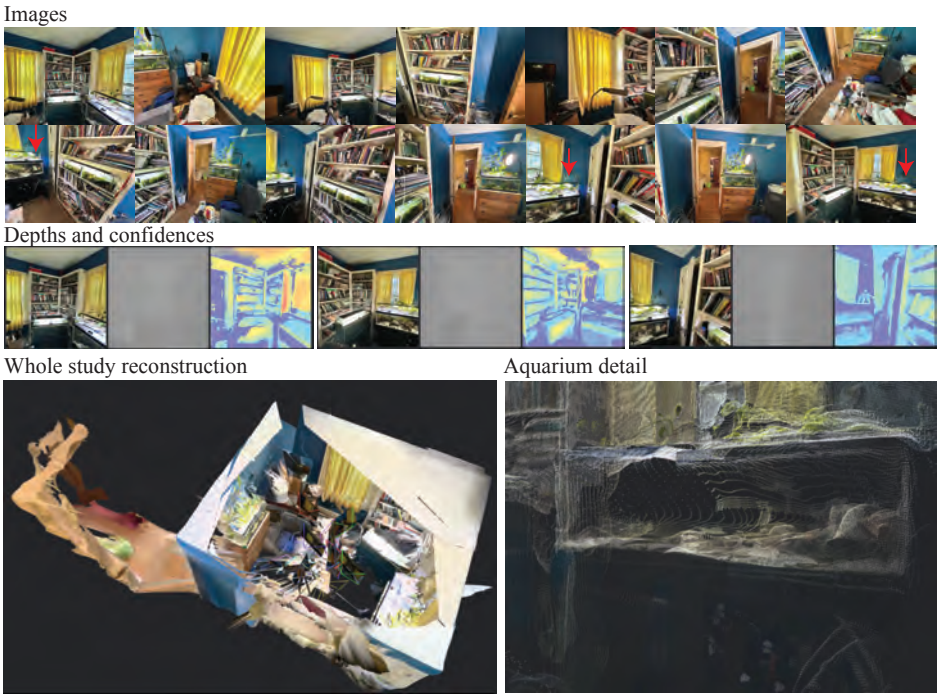


FIGURE 40.5: *An example DUSt3R reconstruction from 14 images (***top row***) of my study. There are three images with per-image depth maps in the* **center row**. **Bottom left** *shows the reconstruction, with camera frustra are shown in strong colors. The view is (roughly) overhead, to show a passage visible through the door. DUSt3R has reconstructed this passage.* **Bottom right**, *shows a detail, showing one of the aquariums which can be seen in some of the images (***red** *arrows). As you would expect, the aquarium glass has not been reconstructed, but the point cloud is relatively sparse on the rocks and there is notable doubling of structures.*

Procedure: 40.8 *Single image depth prediction*

Obtain a single image depth map from $\mathcal{I}$ by forming $\mathcal{P}_1(\mathcal{I}, \mathcal{I})$ and using the third component of the point map.

Procedure: 40.9 *Matching with DUSt3R point maps*

I will refer to the images as A and B (you could match from image one to image two, or vice versa). You wish to find the point in image B that matches $\mathbf{x}_A$ in image A. Query image A's point map at $\mathbf{x}_A$ to find the corresponding 3D point $\mathbf{X}_A$. Find the closest 3D point in image B's point map. The location $\mathbf{x}_B$ in image B corresponding to that point is the match.

This matching procedure offers a proxy evaluation which can be useful. Assume you have two images where you know at least some correct point correspondences across the images. Pass these to DUST3R, recover the predicted correspondences, then compare them to the true correspondences.

Procedure: 40.10 *Obtaining camera intrinsics with DUSt3R point maps*

Write $[x_{ij}, y_{ij}, z_{ij}]^T$ for the 3D position of the point that lands on pixel i, j in image one. This vector is the contents of camera one's point map. Write

$$\max K = \begin{bmatrix} s_x & k & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

for the calibration matrix. Write $\rho(u, C)$ for some cost function which accepts an error u and a confidence weight C (you could use Cu or even u if you're not concerned about robustness). Choose $\mathcal{K}$ by minimizing

$$\sum_{i,j} \rho((\frac{s_x x_{ij} + k y_{ij} + t_x}{z_{ij}} - i)^2 + (\frac{s_y y_{ij} + t_y}{z_{ij}} - j)^2, C^1_{ij}).$$

Procedure: 40.11 *Obtaining camera to camera transformations from DUSt3R, I*

You wish to determine the rotation and transformation from camera A to camera B. Apply DUSt3R to obtain point maps in camera A's frame. Now swap the cameras, and apply DUSt3R to obtain point maps in camera B's frame. Now use the registration methods of Chapter 36.3 to obtain the rotation and transformation from camera A's frame to camera B's frame.

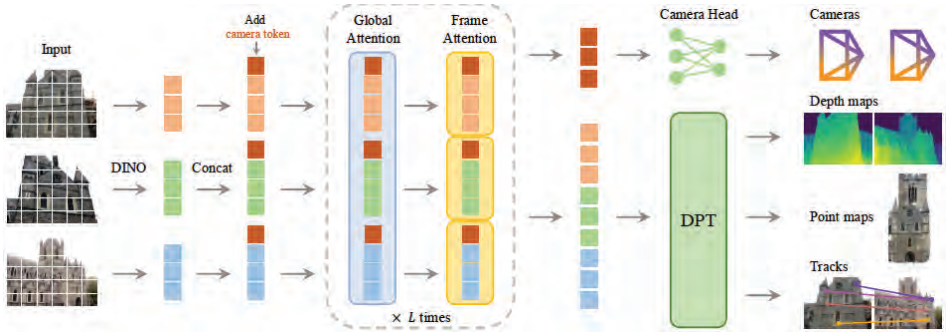


FIGURE 40.6: *VGGT accepts N images; turns each into a sequence of tokens; appends a camera token to each; concatenates the sequences into a frame sequence; and passes the result through a specialized transformer. Each camera token that comes out of the transformer goes to a camera decoder which produces the camera parameters for each frame. The other frame tokens go to a dense image decoder which produces the depth maps, the point maps and the feature maps, as well as an uncertainty map for each depth and point map.*
Image credit *Constructed from Figure 2 of "VGGT: Visual Geometry Grounded Transformer", Jianyuan Wang and Minghao Chen and Nikita Karaev and Andrea Vedaldi and Christian Rupprecht and David Novotny, arXiv, 2025*

Procedure: 40.12 *Obtaining camera to camera transformations from DUSt3R, II*

You wish to determine the rotation and transformation from camera A to camera B. Apply DUSt3R to obtain point maps. Use these point maps to determine a collection of point correspondences from image A to image B. Now use the methods of Chapter 36.3 (PnP, Section 36.3, for example)

40.3.3 DUSt3R and Scene Graphs

You can go considerably further. Assume you have a collection of N images. Make a graph where each vertex is an image. Find all pairs of images that could result in scene graph edges, as in Procedure 38.1. Pass each of these pairs into DUSt3R. If the confidence is sufficient, construct an edge between the two images in the graph. You can now construct estimates of each camera's calibration and of the rotation and translation between the cameras as above. The result is an object very much like the scene graph of Section 38.1.2.

Now choose one camera's coordinate system to be the world coordinate system. You could use the rotation averaging and translation averaging procedures of Section **??** (for example) to obtain a global estimate of where every camera is and where it is pointing. In turn, you can transform the points in the first camera's

coordinate system into the world coordinate system.

40.4 RECONSTRUCTION WITHOUT TEARS

The key significance of DUSt3R is that it shows that one can learn to predict point maps with a method that doesn't explicitly use geometry. But DUSt3R has nuisance features. It accepts only two images. Depth, camera parametrization and relative configuration need to be imputed from the results. Exploiting many observations of the same point requires building some form of scene graph and then applying further optimization. In an ideal world, you would have a learned system that accepts an arbitrary number of images and produces all useful 3D information, without further ado.

40.4.1 VGGT

VGGT accepts a set of N images. For each image, VGGT predicts: six camera extrinsics relative to the world coordinate system; three camera intrinsics; a depth map; a point map, with points in the world coordinate system; and a feature map. The feature map allows you to query with a pixel in one image and obtain a track of the locations in the other images where that pixel appears. VGGT regards the coordinate system of the first camera as the world coordinate system.

VGGT is built with a large transformer encoder, a small decoder for the camera parameters and dense decoders for depth map, point map and feature map. Each of the N images is turned into a sequence of tokens by cutting it into a grid of square patches then passing these through DINOv2 and applying a positional embedding. A camera token is then appended to each image token sequence to make a frame sequence. These frame sequences are concatenated to a single sequence. Each token in this sequence preserves which frame sequence it comes from.

The whole token sequence is then passed to a transformer. This transformer is slightly different from those of Section 36.3. There are two kinds of attention layer. In a global attention layer, every pair of tokens interacts, which means the number of images N will be limited by available resources. In a frame attention layer, interactions occur only between tokens coming from the same frame sequence. These layers are alternated, allowing information to move both across and within frames. The camera tokens go to a camera decoder which produces the camera parameters for each frame. The other frame tokens go to a dense image decoder which produces the depth maps, the point maps and the feature maps. The dense decoder also produces an uncertainty map for each depth and point map. The resulting predictor is then trained with large quantities of 3D annotated data.

The training loss consists of four terms. Write

$$\mathcal{L}_{\text{camera}} + \mathcal{L}_{\text{depth}} + \mathcal{L}_{\text{point map}} + \mathcal{L}_{\text{tracking}}$$

The first three terms are easily predicted from above. The camera loss $\mathcal{L}_{\text{camera}}$ compares the predicted camera parameters with the ground truth using the Huber loss (Section 36.3). The depth loss $\mathcal{L}_{\text{depth}}$ compares predicted depth with ground truth depth, using the uncertainty as in DUSt3R. Unlike DUSt3R, the depth loss

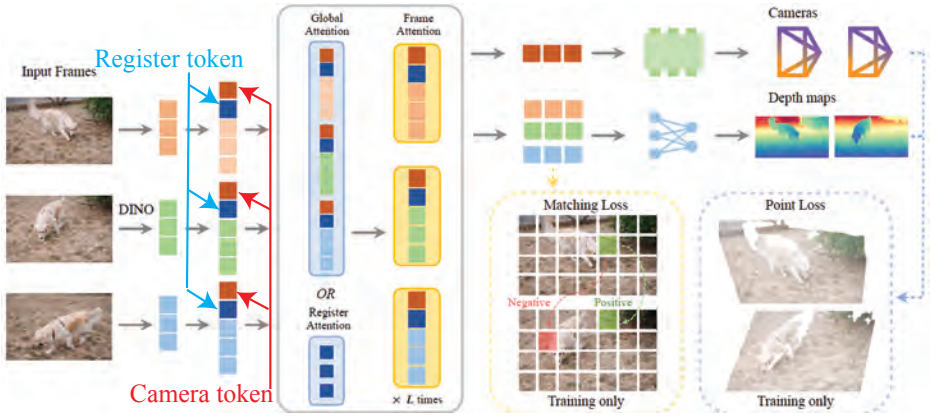


FIGURE 40.7: *You should have noticed global attention has to consume a lot of memory, limiting the number of frames. VGGT-Ω reduces the amount of memory consumed by VGGT by inserting register tokens into each frame token sequence. These register tokens are intended to produce global summary information about a frame. In turn, a quarter of the global attention layers are replaced with register attention layers. In a register attention layer, only the global tokens from each frame interact. While register tokens must somewhat increase the memory required in frame attention layers* **exercises**. *However, register tokens appear to improve the behavior of ViT models in general. Without a register token, the tokens in a ViT must represent both local and global information. The global information tends to be crammed into a few tokens, and this makes the spatial representation incoherent [?, ?].*
Image credit *Constructed from Figure 2 of "VGGT-Ω", Jianyuan Wang and Minghao Chen and Shangzhan Zhang and Nikita Karaev and Johannes Schönberger and Patrick Labatut and Piotr Bojanowski and David Novotny and Andrea Vedaldi and Christian Rupprecht, arXiv, 2026.*

also compares gradient in predicted depth with gradient in ground truth depth. Write D_{ij}^k for the predicted depth at the i, j'th location in the k'th image, $C_{d,ij}^k$ for the depth confidence in the same image, and use an overline to indicate ground truth depth, to get

$$\mathcal{L}_{\text{depth}} = \sum_{ijk} \left[\| C_{d,ij}^k (D_{ij}^k - \overline{D}_{ij}^k) \| + \| C_{d,ij}^k (\nabla D_{ij}^k - \nabla \overline{D}_{ij}^k) \| - \alpha_d \log C_{d,ij}^k \right].$$

The gradient here is a spatial gradient, rather than the gradient with respect to the network parameters **exercises**.

The point loss $\mathcal{L}_{\text{point map}}$ has the same form. Reuse the notation of Section 36.3 to get

$$\mathcal{L}_{\text{point map}} = \sum_{ijk} \left[\| C_{p,ij}^k (X_{ij}^k - \overline{X}_{ij}^k) \| + \| C_{p,ij}^k (\nabla X_{ij}^k - \nabla \overline{X}_{ij}^k) \| - \alpha_p \log C_{p,ij}^k \right].$$



FIGURE 40.8: *VGGT-Ω can predict point maps and depth maps when objects move. The large image on the* **left** *is a view of a point map reconstructed from the images above (and others). Notice how in the reconstruction there are multiple cars on the circular road, far more than in the original images. If a car moves from frame to frame, it is reconstructed in both configurations. Similarly, the large image on the* **right** *is a view of a point cloud reconstructed from the four images on top (and no others). The tennis player has moved from frame to frame, and so appears in each of the four configurations in which they appear in the images.*
Image credit *Constructed from parts of Figure 4 of "VGGT-Ω", Jianyuan Wang and Minghao Chen and Shangzhan Zhang and Nikita Karaev and Johannes Schönberger and Patrick Labatut and Piotr Bojanowski and David Novotny and Andrea Vedaldi and Christian Rupprecht, arXiv, 2026.*

There is one change. In DUSt3R, the predictions and the ground truth are normalized. For VGGT, the ground truth is normalized as in DUSt3R, but the predictions are not – the transformer must learn to predict normalized points.

The fourth loss term is somewhat more intricate. Although the paper calls this a tracking loss, the frames aren't ordered in time, so you should think of it as a correspondence loss. The loss pushes the feature map for each frame to produce features that identify points in other frames that correspond. Identify a set of query points (the paper uses points in the first frame). Now pass the feature maps and the query points into a version of CoTracker (the paper uses CoTracker2). Recall from Section **??** that CoTracker uses a transformer to predict points that correspond to the query points in the other frames. Now compare the predicted points to the ground truth points. Write $\mathbf{y}_{j,i}$ for the ground truth position of the j'th query point in the i'th image and $\hat{\mathbf{y}}_{j,i}$ for the predicted position of that point. The tracking loss is then

$$\sum_{i \in \text{frames}} \sum_{j \in \text{queries}} \| \mathbf{y}_{j,i} - \hat{\mathbf{y}}_{j,i} \|.$$

CoTracker can also predict the visibility of points, and VGGT applies a visibility loss comparing these predictions to ground truth.

The losses are quite heavily correlated. For example, depth predictions in each image together with camera parameter predictions yield point maps; similarly, point maps and camera parameters yield depth predictions; and so on **exercises**.



FIGURE 40.9: *An example VGGT-Ω reconstruction from 14 images (shown below) of my study. The camera frustra are shown in strong colors. Note the passage visible through the door, reconstructed in Figure 40.5, is now absent. More information in text.*

Experience teaches that, at least in the case of 3D prediction, using losses like this improves performance. For example, even if you did not wish to construct point correspondences, it is a good idea to produce a feature map and evaluate a tracking loss.

40.4.2 VGGT-Ω

VGGT-Ω uses a number of innovations to reduce the amount of GPU memory required by a set of frames. This means you can use more frames during both inference and training without increasing your GPU resources. Part of the difficulty with VGGT is the number of attention terms required by the cross frame encoder. Figure 40.13 sketches the innovations used to control this difficulty. Decoding into multiple dense maps (the depth map; the point map; the feature map) must also consume a lot of memory. VGGT-Ω deals with this by decoding into only camera parameters and depth map. The depth map loss can then be computed as before. At training time, point maps are predicted using the predicted camera parameters and the depth map (Figure 40.13) **exercises**. The point map loss is then applied to these predicted point maps.

The tracking loss is quite different from VGGT's tracking loss, because VGGT-Ω does not predict a feature map. Instead, training data is used to identify tokens that correspond across frames. The loss should require that tokens that correspond have similar representations and tokens that don't correspond generally have different representations (you should compare this with a contrastive loss, Section 23.3). Write $s(i, j)$ for the cosine similarity between tokens i and j, $\sigma(u)$ for the sigmoid function of u, $\mathcal{P}$ for the set of pairs of tokens that correspond and $\mathcal{N}$ for the set of



FIGURE 40.10: *A detail from the VGGT-Ω reconstruction of Figure 40.9, showing the aquarium which can be seen in some of the images (***red** *arrows, below). As you would expect, the aquarium glass has not been reconstructed, but the point cloud is relatively sparse on the rocks and there is notable doubling of structures. Compare with the reconstruction of Figure 40.10. More details in text.*

pairs that do not correspond. The loss is

$$\frac{1}{|\mathcal{P}|}\sum_{i,j\in\mathcal{P}}\left[-\log\sigma(s(i,j))\right]+\frac{1}{|\mathcal{N}|}\sum_{u,v\in\mathcal{N}}\left[-\log\left(1-\sigma(s(i,j))\right)\right].$$

Constructing pairs of tokens that correspond requires some care. Each token is associated with an image patch. Assume there are point clouds in the training data. Each 3D point projects to multiple different image patches, and so can be associated with their token. If two image patches from different frames have enough 3D points in common, then they correspond and belong in $\mathcal{P}$. The set $\mathcal{N}$ should contain patches that simply could not correspond. Use all pairs of patches that have sufficiently different appearance. Further, use patches where one is too far away from the epipolar line predicted by the center of the other.

40.4.3 DepthAnythingV3

The systems I have described up to here predict camera extrinsics in a fixed world coordinate system. This means you must predict rotation matrices. But ensuring the orthonormality constraint is respected can be tricky. An alternative is to predict a ray map.

DepthAnythingV3 [**?**] predicts a depth map and a ray map for each image in a set. Prediction is by a transformer with a dense decoder. Both transformer and decoder differ in detail from those used by VGGT and VGGT-Ω, but the details are out of scope.

Predicting depths or rays at sky pixels causes problems because the depth there is undefined. In the DUSt3R and VGGT models, this issue is controlled by



FIGURE 40.11: *Errors and omissions in ground truth data present a significant problem in training 3D predictors. DepthAnythingV3 makes a convincing argument that using a very large quantity of data does not control this difficulty. On the* **bottom left**, *Figure 19.6, reproduced here for your convenience. This shows an example of an RGB image registered to a depth map obtained with a Kinect device. The black points correspond to locations where the Kinect could not determine depth. On the* **right**, *further examples of problems in training datasets. Notice two are constructed using ColMap, and one is constructed using Lidar. In each case, there remain points where depth is not satisfactorily computed.*
Image credit *Constructed from parts of Figure 4 of "Depth Anything 3: Recovering the visual space from any views", Haotong Lin and Sili Chen and Jun Hao Liew and Donny Y. Chen and Zhenyu Li and Guang Shi and Jiashi Feng and Bingyi Kang, arXiv, 2025.*
Image credit: *The images on the* **left** *come from the very well known NYUV2 dataset, published at* `https://cs.nyu.edu/~fergus/datasets/nyu_depth_v2.html`.

the confidence map. DepthAnythingV3 uses the alternative of predicting a sky mask and an object mask, both trained from ground truth. These are then used to mask the losses.

DepthAnythingV3 uses a depth gradient loss, as above. There are four other losses. A local-global loss consists of two terms. The first compares point predictions with ground truth, averaged over all points. The second forces local point predictions to be accurate. These losses require optimization during training because the prediction may be at a different scale and translation from the ground truth. DepthAnythingV3 uses an efficient estimator, originating with [**?**].

A normal loss compares predicted surface normal and ground truth surface normal at each pixel in two ways. The first uses a surface normal computed from the depth map at each pixel, and the second uses a smoothed estimate of the normal. Smoothing is by a weighted sum of nearby normal estimates. Two other losses control errors in predicting the sky mask and the object mask respectively.

DepthAnythingV3 uses two strategies to control problems caused by unre-



FIGURE 40.12: *DepthAnythingV3 is trained using pseudo-labelled data. Synthetic images don't require correction, but real images where the depth displays effects like those of Figure 40.11 are passed to a teacher model. This accepts an image and produces a depth estimate, which is then registered to that of the original depth. The result replaces the noisy depth. The teacher model itself is trained only on synthetic data, so that its boundary predictions can be crisp.* **Top** *row shows example input images;* **middle** *row shows depth maps predicted from those images using a version of DepthAnythingV3 trained without the teacher model (redder is closer, bluer further away); and* **bottom** *row shows depth maps predicted using a version of DepthAnythingV3 trained with the teacher model. Notice the significant improvement in predictions around fine spatial details.* Image credit *Constructed from Figure 8 of "Depth Anything 3: Recovering the visual space from any views", Haotong Lin and Sili Chen and Jun Hao Liew and Donny Y. Chen and Zhenyu Li and Guang Shi and Jiashi Feng and Bingyi Kang, arXiv, 2025.*

liable training data (Figure 40.11). First, training uses a substantial amount of synthetic data. Second, a teacher model, trained entirely on synthetic data, is used to produce improved depth estimates for the real image data. The teacher model has significant effects on results (Figure **??**).

40.4.4 Permutation Invariance and Π^3

The point maps that DUSt3R produces depend on which image is first and which image is second, because the point maps are in the first image's coordinate system. Similarly, the reconstructions that VGGT, VTTG-Ω and DepthAnythingV3 produce are in the first image's coordinate system. If you reorder the images you pass into any of these systems and the first image changes, you must get a different reconstruction. This is because the new reconstruction will be with respect to a new coordinate system.

Ideally, the new reconstruction will be within a rotation, translation and scale of the original one. This is because (a) the reconstructions are up to scale and (b) the change of coordinate system is a rotation and a translation. But nothing in the

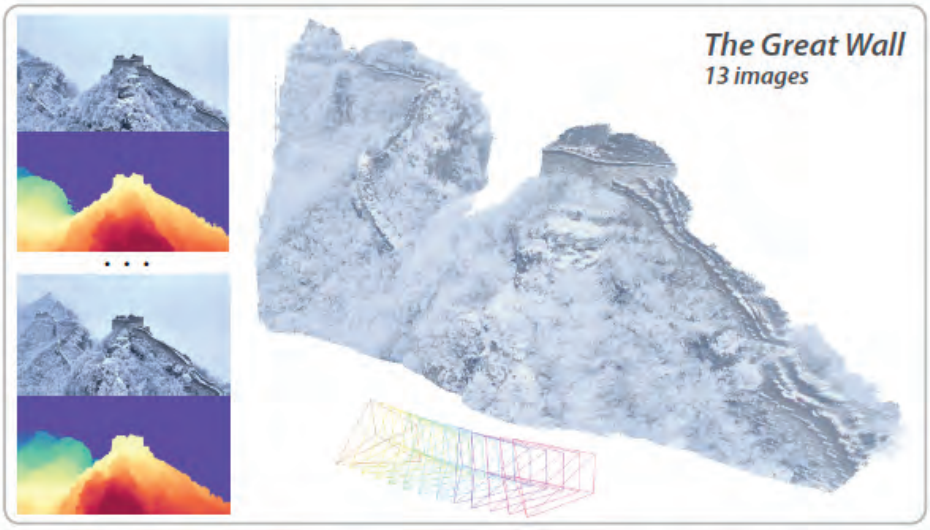


FIGURE 40.13: *DepthAnythingV3 can produce point clouds of very large objects from relatively few images; here a portion of the Great Wall is reconstructed from 14 images.* Image credit *Constructed from part of Figure 9 of "Depth Anything 3: Recovering the visual space from any views", Haotong Lin and Sili Chen and Jun Hao Liew and Donny Y. Chen and Zhenyu Li and Guang Shi and Jiashi Feng and Bingyi Kang, arXiv, 2025.*

construction or training of the system guarantees this property. Worse, the fact that there is a "special image" – the first one – suggests the system should treat at least one image differently from others. Finally, experiment strongly suggests that reordering the images fed to any of these systems produces reconstructions that have different geometries – they are not within a rotation, translation and scale of one another (Figure 40.14; [**?**]).

The desirable property here is *permutation equivariance* – if you reorder the images you pass into the system, what comes out is simply reordered. Π^3 is a system that achieves permutation equivariance without changes to the transformer architecture. Π^3 handles tokenization, point maps and training losses with care. Positional embeddings attached to tokens can encode where a token is within a frame, but cannot encode which frame in the sequence the token is in. If they did, the tokenization would change when you reordered the input frames. In turn, the output would change. Point maps cannot be in a world coordinate system, because that would depend on a reference frame. Instead, the point maps reported for each frame are reported *in that frame's coordinate system.* This means constructing a point cloud requires using each frame's camera extrinsics.

Losses require some care, too You cannot compare the camera extrinsic parameters to ground truth, because the ground truth depends on a choice of reconstruction coordinate system. Instead, require that camera-to-camera transformations are consistent with ground truth. Write $\mathcal{T}_i$ for the transformation taking a point

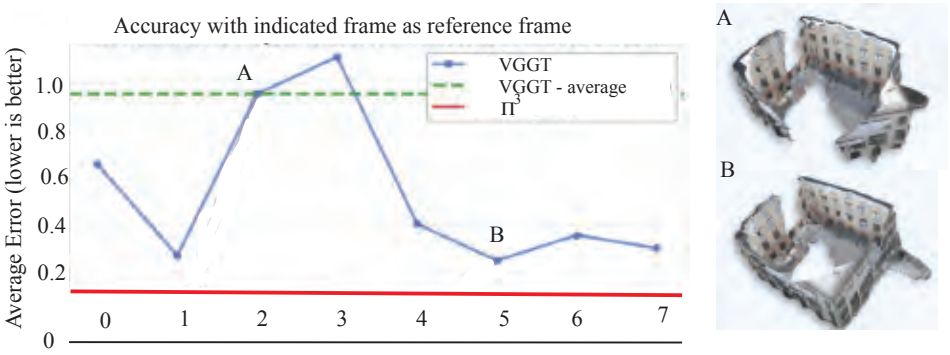


FIGURE 40.14: *Rearranging the frames passed to VGGT so that a different frame is used as a reference frame can produce real changes in the reconstruction. In the graph on the* **left**, *the horizontal axis shows which frame in a sequence is used as the reference frame (take the sequence and move that frame to the first position). The vertical axis shows reconstruction error against ground truth for a test sequence. On the* **right**, *two distinct reconstructions of the same set of images, obtained with different initial frames. Notice how the choice of first frame has a real effect on the resulting reconstruction.* Image credit *Constructed from parts of Figure 2 of "*Π^3*:Permutation-equivariant Visual Geometry Learning", Yifan Wang, Jianjun Zhou, Haoyi Zhu, Wenzheng Chang, Yang Zhou, Zizun Li, Junyi Chen, Jiangmiao Pang, Chunhua Shen, Tong He, ICLR, 2026*

in the world coordinate system to a point in the i'th camera's coordinate system (which you can obtain from the i'th camera's extrinsics **exercises**). Then the transformation from the i'th camera's coordinate system to the j'th camera's coordinate system is $\mathcal{T}_j \circ \mathcal{T}_j^{-1}$. Write $\hat{\mathcal{T}}_i$ for the i'th camera extrinsics reported by the system. Rather than comparing $\mathcal{T}_i$ to $\hat{\mathcal{T}}_i$, you must compare

$$\mathcal{T}_i \circ \mathcal{T}_j^{-1} \text{ to } \hat{\mathcal{T}}_i \circ \hat{\mathcal{T}}_j^{-1}$$

for each pair i, j. Notice that if $\left\{\hat{\mathcal{T}}_1, \ldots, \hat{\mathcal{T}}_N\right\}$ minimizes this loss, then so will $\left\{\hat{\mathcal{T}}_1 \circ \mathcal{M}, \ldots, \hat{\mathcal{T}}_N \circ \mathcal{M}\right\}$, where $\mathcal{M}$ is some arbitrarily chosen rotation, translation and scale **exercises**. Equivalently, the system is (essentially) allowed to choose the coordinate system in which it reconstructs.

40.4.5 Online Prediction and CUT3R

Each of the systems above accepts a fixed set of images and produces a geometric representation for them. Under some circumstances, you want to pass a stream of images into a model and obtain a reconstruction at any time. Write $\mathcal{I}_t$ for the image at t (which you can think of as time, but doesn't absolutely have to be). You would like to be able to obtain a reconstruction $\mathcal{R}_{\ldots\tau}$ which incorporates all information up to the query time τ. You could achieve this by storing all images up to τ, then passing the whole lot into one of the previous models, but the disadvantages of this

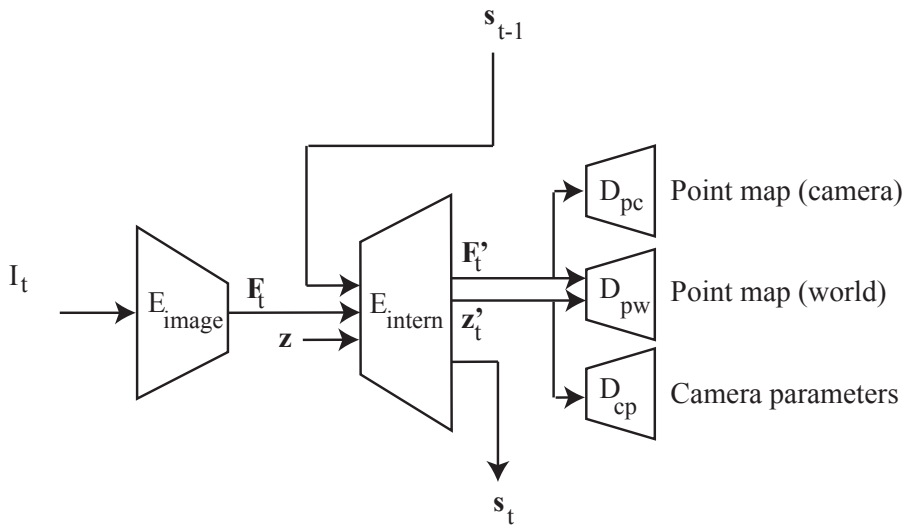


FIGURE 40.15: *CUT3R accepts a stream of images and produces a stream of state representations together with a point map for each image in the camera's frame, a point map for that image in a world frame, and an estimate of camera parameters for the frame. The image $\mathcal{I}_t$ at time t is encoded using a fixed VIT encoder into a representation $\mathbf{F}_t$. A further encoder accepts that representation, the state vector $\mathbf{s}_{t-1}$, and a learnable camera token $\mathbf{z}$, and produces a modified image encoding $\mathbf{F}'_t$, an updated state vector $\mathbf{s}_t$, and a camera token $\mathbf{z}'_t$. Finally, $\mathbf{F}'_t$ is decoded to a point map in the camera's frame; $\mathbf{F}'_t$ and $\mathbf{z}'_t$ are decoded to a point map in the world coordinate system; and $\mathbf{z}'_t$ is decoded into a set of camera parameters. The state representations can be queried to produce a variety of representations, details in text.*

approach should be fairly obvious **exercises**.

The alternative is that the model maintains some form of state. CUT3R is a model that does this. Figure 40.15 show the overall structure of the model. CUT3R is trained to predict a point map in the camera's coordinate system (effectively a depth map); a point map in the world coordinate system; and a set of camera parameters. It does so for each frame.

In principle, you could obtain a point cloud from N images using CUT3R by passing in the images, collecting each point map in the world coordinate system as it appears, then forming the union of the point sets. Doing so rather misses what is important about the state. $\mathbf{s}_{t-1}$ must store a 3D representation of all that has been seen up to time t, otherwise the decoders could not reconstruct a point map in camera t's frame. In turn, it is possible to decode $\mathbf{s}_t$ in interesting ways.

For example, assume you have an encoder that takes a ray map in the world coordinate system to a representation comparable with $\mathbf{F}_t$. You could then query the state vector as in Figure 40.16. Querying the state vector with a ray map that covers regions of space that haven't been seen produces quite reasonable extrapo-

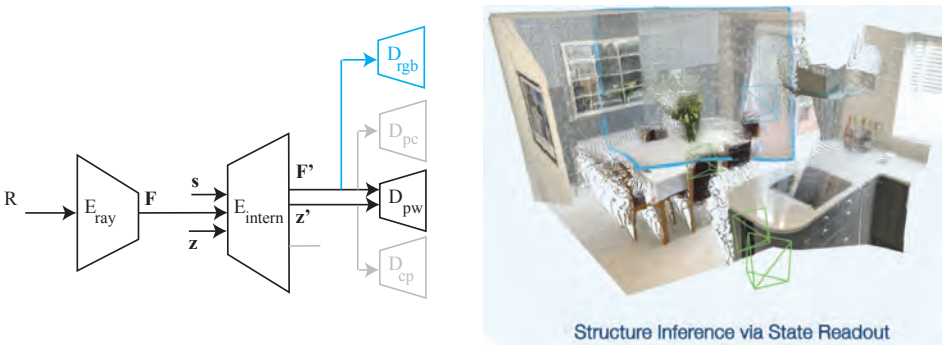


FIGURE 40.16: *Given an encoder that transforms a ray map R into a representation compatible with CUT3R's image representation, you can recover a point map from the state vector. Do so (***left***) by reusing some (***black***) and ignoring other (***gray***) components of the system of Figure 40.15. A further decoder (***blue***) provides the color of each point on the point map. On the* ***right****, a point map queried in this way (***light blue***) superimposed on a CUT3R reconstruction.* Image credit *Constructed using part of Figure 3 of "Continuous 3D Perception Model with Persistent State", Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A. Efros, Angjoo Kanazawa, CVPR, 2025.*

lates (Figure **??**).

The process of Figure 40.16 will yield different results if you use different state vectors, because $\mathbf{s}_t$ has seen only images up to t. You would normally use the last state vector. More generally, the state vector is *causal* – it cannot contain information from the future. For many kinds of reconstruction this is a real disadvantage, because frames in "the future" may quite strongly constrain reconstructions at time t. You can avoid this difficulty by passing a sequence through CUT3R, capturing the state vector at the final frame, then passing the sequence through CUT3R again using this state vector as an initial state (Figure **??**).

CUT3R is trained with sets of images together with known depth and point maps. There are three encoders (E_{image}, E_{internal} and E_{ray}). There are several decoders: D_{pc} produces a point map in the camera frame; D_{pw} produces a point map in the world frame; D_{pc} produces the camera parameters; and D_{rgb} predicts the color of points when the encoded object is a raymap. As for earlier models, losses penalize incorrect prediction of point maps (with a confidence term, as in Section 40.3.1) and camera parameters. If metric depths are known for the set of images, some frames are randomly replaced with raymaps and the raymap encoder is applied to these raymaps so as to train the raymap encoder. Such frames are also used to train the color decoder.

40.4.6 D4RT

Representing a 3D scene – where nothing moves – as a collection of per image point maps must have some serious redundancies, because the same point will appear

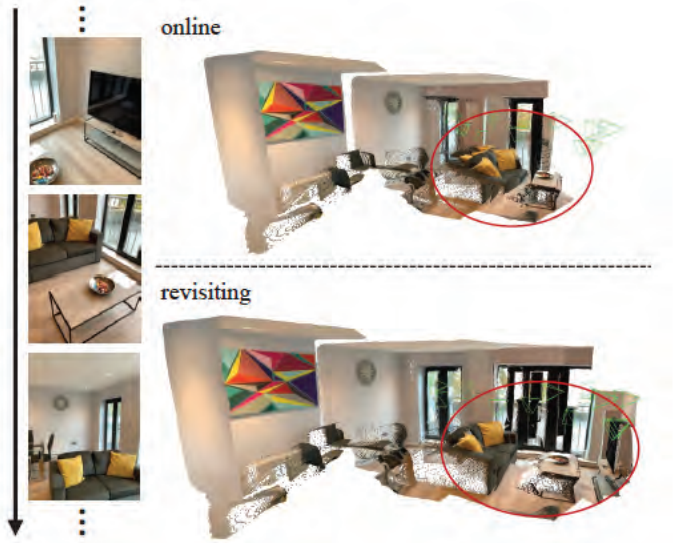


FIGURE 40.17: *The consequences of using a causal state vector. The* **left** *shows the images. The* **top right** *shows a reconstruction of a region mostly visible in early frames (match the television in the first frame), using a purely causal reconstruction. Future images haven't affected this reconstruction, and some information is missing or wrong (look at the* **red** *ellipse). If you capture the last state vector, then re-run CUT3R with that as initial state, you get the reconstruction on the* **bottom right**. *Notice the considerable improvement in detail that results* Image credit *Constructed using part of Figure 5 of "Continuous 3D Perception Model with Persistent State", Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A. Efros, Angjoo Kanazawa, CVPR, 2025.*

in several different point maps. However, you can usually tell what point in one per image point map corresponds to what in another. Just look for the nearest neighbor. If the scene is a 4D scene – where some objects are moving – there are serious difficulties. There is no elementary way to tell what point in one point map corresponds to what in another. Nearest neighbors isn't good enough because the point may have moved. You would need a more complex matching procedure.

D4RT implements an interesting alternative, which (as of writing) applies only to video. Video sequences have the advantage that generally there will be many correspondences between a frame and the next frame. In turn, reconstructions may not need very long range matches. D4RT encodes an entire video sequence into a single code (which is necessarily large). D4RT provides a lightweight decoder that allows you to query where a particular point in a particular frame would be at a new time in the coordinate system of a specified camera. Write u_s, v_s for the coordinates of the point in the frame of interest (at time t_s); t_q for the query time (the time when you want the location of the point); and t_{cam} for the time stamp of



FIGURE 40.18: *CUT3R can predict geometry for locations that have not been seen – these are extrapolates. For four cases, the* **top** *row shows input images, shown to CUT3R. The* **upper middle** *row shows images obtained from novel viewpoints. These images are not shown to CUT3R, but the viewpoints are used to produce raymaps which then query the model. The* **lower middle** *row shows a 3D view of CUT3R's reconstruction from images not including the* **upper middle** *row. Finally, CUT3R is queried using the raymaps corresponding the upper middle row, to produce the extrapolated reconstructions of the* **bottom row**. *The* **green** *arrows point to view frusta associated with the raymaps.* Image credit *Constructed using part of Figure 5 of "Continuous 3D Perception Model with Persistent State", Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A. Efros, Angjoo Kanazawa, CVPR, 2025.*

the camera coordinate system in which you want the point reported. Then D4RT's encoder accepts the tuple $(u_s, v_s, t_s, t_q, t_{\text{cam}})$ and produces the 3D point location.

This is much more powerful than it might seem at first glance. A number of apparently different reconstructions come from building up queries out of this basic component. Write T for the total number of frames, so t runs from $1 \dots T$ and each frame is $U \times V$. Then you obtain:

- a **point track** from a series of queries $(u_s, v_s, 1, 1, 1) \dots (u_s, v_s, 1, T, T)$;
- a **point cloud** from a collection of queries $(u, v, t_a, t_b, t_{\text{cam}})$, where $u \in \{1 \dots U\}$, $v \in \{1 \dots V\}$, $t_a \in \{1 \dots T\}$, and $t_b \in \{1 \dots T\}$;
- **per image depth maps** from a collection of queries (u, v, t_a, t_a, t_a), where $u \in \{1 \dots U\}$, $v \in \{1 \dots V\}$, $t_a \in \{1 \dots T\}$, and you then recover depth from

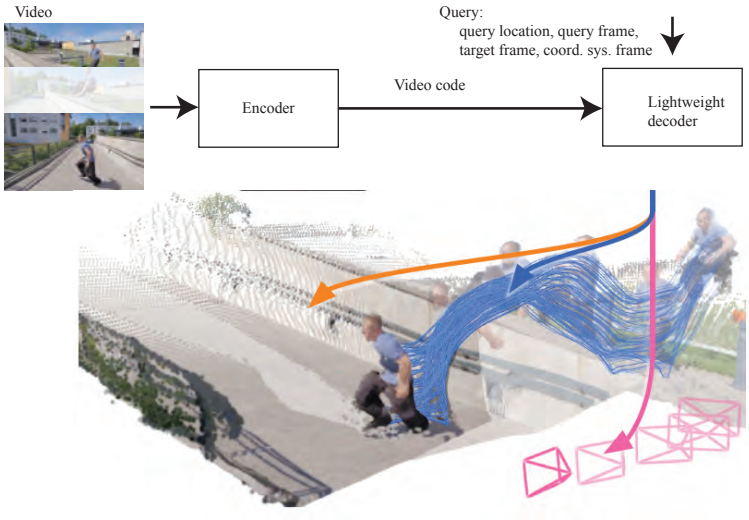


FIGURE 40.19: *D4RT encodes an entire video into a latent representation, which is necessarily large. A lightweight encoder then accepts some query parameters identifying a point (where it is in a frame; which frame the location refers to) and others identifying a reconstruction (the time at which it should be reconstructed; the frame of the camera in whose coordinate system it should be reconstructed) and produces the 3D location of that point. Various combinations of queries can be used to yield, among other things, a point cloud (***orange*** *arrow); point tracks (****blue*** *tracks); or camera parameters (****pink*** *frusta).* Image credit *Constructed using parts of Figure 1 of "Efficiently Reconstructing Dynamic Scenes One D4RT at a Time", Chuhan Zhang, Guillaume Le Moing, Skanda Koppula, Ignacio Rocco, Liliane Momeni, Junyu Xie, Shuyang Sun, Rahul Sukthankar, Joëlle K. Barral, Raia Hadsell, Zoubin Ghahramani, Andrew Zisserman, Junlin Zhang, Mehdi S. M. Sajjadi, CVPR 2026*

the 3D point;

- the change in **camera extrinsics** from frame t_a to t_b by choosing a set of points $\mathcal{S}$ in frame t_a, forming a collection of queries (u, v, t_a, t_a, t_a) and from a collection of queries (u, v, t_a, t_a, t_a) and (u, v, t_a, t_b, t_b), where $(u, v) \in \mathcal{S}$, and then registering the corresponding point sets (easy, because you have correspondence **exercises**);
- the **camera path** by recovering the change in extrinsics from frame to frame;
- the **camera intrinsics** for frame t_a by choosing a set of points $\mathcal{S}$ in frame t_a, forming a collection of queries (u, v, t_a, t_a, t_a) and then fitting an intrinsic model.

Some care is required with the point cloud query to avoid reconstructing many tracks for the same point in 3D. This is because a point you see in frame 1 very

likely appears in frame 2 as well. If you query the corresponding point in both frames, you will have a redundant point cloud. But if you query only one frame, you may find there are some points you haven't reconstructed. Camera moves may mean that a point you see in an earlier or later frame doesn't appear in the frame you are working with. Worse, an object may move into or out of view.

You can get a good point cloud with a straightforward procedure of marking points that you have seen. Choose some pixel in some frame that hasn't been marked. Reconstruct its path through all frames. In every frame where it is visible, mark the pixel where the point appears. Continue this procedure until every pixel in every frame has been marked.

40.5 RESOURCES

There is a substantial volume of open source software. I have included some systems that I did not describe in detail.

Resource pointers: 40.1 *Viser*

Viser is a visualization library, widely used to display products of reconstruction. The authors provides code and examples at `https://github.com/nerfstudio-project/viser`

Resource pointers: 40.2 *DUST3R*

Naver labs provide an official implementation of DUST3R and model weights at `https://github.com/naver/dust3r`.

Resource pointers: 40.3 *Spann3R*

Spann3R accepts either a video or a set of images of a 3D scene and produces a point map. Spann3R can accept more than two images, and the points are in a global coordinate system. There is a MonST3R page at `https://hengyiwang.github.io/projects/spanner` giving paper, examples and links. The authors provides code and pretrained models at `https://github.com/HengyiWang/spann3r`

Resource pointers: 40.4 *VGGT*

There is a VGGT page at `https://vgg-t.github.io`, giving paper, examples and links. Facebook research provides code and pretrained models at `https://github.com/facebookresearch/vggt`.

Resource pointers: 40.5 *VGGT-Ω*

There is a VGGT-Ω page at `https://vggt-omega.github.io`, giving paper, examples and links. Facebook research provides code and pretrained models at `https://github.com/facebookresearch/vggt-omega`.

Resource pointers: 40.6 *DepthAnythingV3*

There is a DepthAnything V3 page at `https://depth-anything-3.github.io`, giving paper, examples and links. Bytedance provides code and pretrained models at `https://github.com/ByteDance-Seed/depth-anything-3`

Resource pointers: 40.7 *CUT3R*

There is a CUT3R page at `https://cut3r.github.io`, giving paper, examples and links. The authors provides code and pretrained models at `https://github.com/CUT3R/CUT3R`.

Resource pointers: 40.8 *MonsT3R*

MonST3R accepts a video of a 4D scene and produces a time-varying point cloud. There is a MonST3R page at `https://monst3r-project.github.io`, giving paper, examples and links. The authors provides code and pretrained models at `https://github.com/junyi42/monst3r`

Resource pointers: 40.9 *D4RT*

There is a D4RT page at `https://d4rt-paper.github.io` giving paper and examples. I am not aware of any open code or model, however.

Resource pointers: 40.10 *Datasets, I*

Most models are now trained on a number of different datasets. Here I reproduce a list of datasets given on the CUT3R github page, for your convenience. The list is in two parts (otherwise it won't fit on the page).

- ARKitScenes https://github.com/apple/ARKitScenes
- BlendedMVS https://github.com/YoYo000/BlendedMVS
- CO3Dv2 https://github.com/facebookresearch/co3d
- MegaDepth https://www.cs.cornell.edu/projects/megadepth/"rel="nofollow
- ScanNet++ https://kaldir.vc.in.tum.de/scannetpp/"rel="nofollow
- ScanNet http://www.scan-net.org/ScanNet/"rel="nofollow
- WayMo Open dataset https://github.com/waymo-research/waymo-open-dataset
- WildRGB-D https://github.com/wildrgbd/wildrgbd/
- Map-free https://research.nianticlabs.com/mapfree-reloc-benchmark/dataset"rel="nofollow
- TartanAir https://theairlab.org/tartanair-dataset/"rel="nofollow
- UnrealStereo4K https://github.com/fabiotosi92/SMD-Nets
- Virtual KITTI 2 https://europe.naverlabs.com/research/computer-vision/proxy-virtual-worlds-vkitti-2/"rel="nofollow
- 3D Ken Burns https://github.com/sniklaus/3d-ken-burns.git
- BEDLAM https://bedlam.is.tue.mpg.de/"rel="nofollow
- COP3D https://github.com/facebookresearch/cop3d

Resource pointers: 40.11 *Datasets, II*

Most models are now trained on a number of different datasets. Here I reproduce a list of datasets given on the CUT3R github page, for your convenience. The list is in two parts (otherwise it won't fit on the page).

- DL3DV `https://github.com/DL3DV-10K/Dataset`
- Dynamic Replica `https://github.com/facebookresearch/dynamic_stereo`
- EDEN `https://lhoangan.github.io/eden/"rel="nofollow`
- Hypersim `https://github.com/apple/ml-hypersim`
- IRS `https://github.com/HKBU-HPML/IRS`
- Matterport3D `https://niessner.github.io/Matterport/"rel="nofollow`
- MVImgNet `https://github.com/GAP-LAB-CUHK-SZ/MVImgNet`
- MVS-Synth `https://phuang17.github.io/DeepMVS/mvs-synth.html"rel="nofollow`
- OmniObject3D `https://omniobject3d.github.io/"rel="nofollow`
- PointOdyssey `https://pointodyssey.com/"rel="nofollow`
- RealEstate10K `https://google.github.io/realestate10k/"rel="nofollow`
- SmartPortraits `https://mobileroboticsskoltech.github.io/SmartPortraits/"rel="nofollow`
- Spring `https://spring-benchmark.org/"rel="nofollow`
- Synscapes `https://synscapes.on.liu.se/"rel="nofollow`
- UASOL `https://osf.io/64532/"rel="nofollow`
- UrbanSyn `https://www.urbansyn.org/"rel="nofollow`
- HOI4D `https://hoi4d.github.io/"rel="nofollow`

40.6 YOU SHOULD

40.6.1 remember these facts:

40.6.2 remember these procedures:

Focal point from ray map . . . 738
Camera rotation and intrinsics from ray map . . . 739
Evaluating per-image depth maps . . . 739
Chamfer distance, v1 . . . 740
Chamfer distance, v2 . . . 740
Relative translation accuracy, or RTA . . . 741
Relative Rotation Accuracy, or RRA . . . 742
Single image depth prediction . . . 746
Matching with DUSt3R point maps . . . 747
Obtaining camera intrinsics with DUSt3R point maps . . . 747
Obtaining camera to camera transformations from DUSt3R, I . . . 747
Obtaining camera to camera transformations from DUSt3R, II . . . 748

40.6.3 exploit these resources:

Viser . . . 763
DUST3R . . . 763
Spann3R . . . 763
VGGT . . . 764
VGGT-Ω . . . 764
DepthAnythingV3 . . . 764
CUT3R . . . 764
MonsT3R . . . 764
D4RT . . . 764
Datasets, I . . . 765
Datasets, II . . . 766

40.6.4 be able to:

- SOMETHIGN

40.6.5 read these materials:

- "DUSt3R: Geometric 3D Vision Made Easy," Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii and Jerome Revaud, CVPR, 2024
- "VGGT: Visual Geometry Grounded Transformer", Jianyuan Wang and Minghao Chen and Nikita Karaev and Andrea Vedaldi and Christian Rupprecht and David Novotny, CVPR, 2025
- "VGGT-Ω", Jianyuan Wang and Minghao Chen and Shangzhan Zhang and Nikita Karaev and Johannes Schönberger and Patrick Labatut and Piotr Bojanowski and David Novotny and Andrea Vedaldi and Christian Rupprecht, CVPR, 2026.

- "Continuous 3D Perception Model with Persistent State", Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A. Efros, Angjoo Kanazawa, CVPR, 2025.
- "Efficiently Reconstructing Dynamic Scenes One D4RT at a Time", Chuhan Zhang, Guillaume Le Moing, Skanda Koppula, Ignacio Rocco, Liliane Momeni, Junyu Xie, Shuyang Sun, Rahul Sukthankar, Joëlle K. Barral, Raia Hadsell, Zoubin Ghahramani, Andrew Zisserman, Junlin Zhang, Mehdi S. M. Sajjadi, CVPR 2026

EXERCISES

QUICK CHECKS

40.1. Section 40.1.2 has "For an ideal camera, the first three components of the map should have the same value at each location. " Explain.

40.2. How good do you expect the focal point estimate from the ray map of Procedure to be? Why?

40.3. Why do cameras have lenses?

40.4. For smaller Δt, images will be darker; for larger Δt, moving objects may have blurred outlines. Why?

40.5. A common form of camera response function is $\text{Intensity}(P\Delta t) = Ce^{\gamma P \Delta t}$. Do you expect γ to be positive or negative? Why?

40.6. Why are mosaiced color CCD cameras more common than multiccd cameras?

40.7. You want to increase the contrast in an image by a pointwise image transformation that (a) makes dark pixels darker *and* (b) makes bright pixels brighter. Why will $f(I) = Ce^{\gamma I}$ not achieve this? Sketch a transformation that will.

40.8. Why is it hard to build a stereo camera that is (a) small and (b) good at measuring large depths?

40.9. What happens in a scanning time of flight sensor if the mirror moves slowly and objects in the scene move quickly?

40.10. An object is 30m away from a scanning time of flight sensor. How long does it take for the pulse to travel from camera to object to sensor? You can take the speed of light to be 3×10^8 meters per second.

LONGER PROBLEMS

40.11. Assume you have an image obtained in a camera with intrinsic matrix $\mathcal{K}$, and the point map is represented in the camera's coordinate system (focal point at the origin, image plane at $z = f$, and so on). Write d_{ij} for the depth to the i, j'th location in the image. If the point map for the image was correct, show the i, j'th location would contain

$$\mathcal{K}^{-1} \begin{bmatrix} id_{ij} \\ jd_{ij} \\ d_{ij} \end{bmatrix}.$$

40.12. Assume you have two images. You know the intrinsic matrices of the cameras ($\mathcal{K}_1$ and $\mathcal{K}_2$ respectively). Choose a world coordinate system aligned with the first camera (focal point at the origin, image plane $z = f$ and so on). The second camera is obtained by rotating the first by $\mathcal{R}$, then translating it by $\mathbf{t}$, then changing the calibration.

(a) Show a point at

$$\mathbf{u}'(i,j) = \mathcal{K}_2^{-1} \begin{bmatrix} id_{2,ij} \\ jd_{2,ij} \\ d_{2,ij} \end{bmatrix}$$

in the second camera's coordinate system projects to the i, j'th pixel in the second camera.

(b) Show that this point lies at $\mathcal{R}^T(\mathbf{u}'(i,j) - \mathbf{t})$ in the first camera's coordinate system.

(c) Show that, if the point map for the second camera is predicted using the first camera's coordinate system, the prediction at (i, j) in the second camera should be

$$\mathcal{R}^T(\mathcal{K}_2^{-1}\begin{bmatrix} id_{2,ij} \\ jd_{2,ij} \\ d_{2,ij} \end{bmatrix}\mathbf{u}'(i,j) - \mathbf{t}).$$

40.13. You wish to recover camera intrinsics and rotation from a ray map. Assume that the focal point estimate is correct. In world coordinates, the camera is $\mathcal{K}\left[\mathcal{R}| - \mathcal{R}\hat{\mathbf{f}}\right]$. The i, j'th 3D point projects to (x_{ij}, y_{ij}) in the image.

(a) Write the 3D point in world coordinates using homogenous coordinates, apply the camera, and show that you obtain

$$\begin{bmatrix} x_{ij} \\ y_{ij} \\ 1 \end{bmatrix} \equiv \mathcal{K}\left[\mathcal{R}| - \mathcal{R}\hat{\mathbf{f}}\right]\begin{bmatrix} \hat{\mathbf{f}} + D_{ij}\mathbf{d}_{ij} \\ 1 \end{bmatrix}.$$

(b) Show the previous expression simplifies to

$$\begin{bmatrix} x_{ij} \\ y_{ij} \\ 1 \end{bmatrix} \equiv \mathcal{K}\mathcal{R}\mathbf{d}_{ij} \equiv \mathcal{H}\mathbf{d}_{ij}.$$

(c) How would you obtain the homography?

(d) How would you obtain the intrinsics and rotation?