Assignment 4

Model Fitting + Stereo/MVS

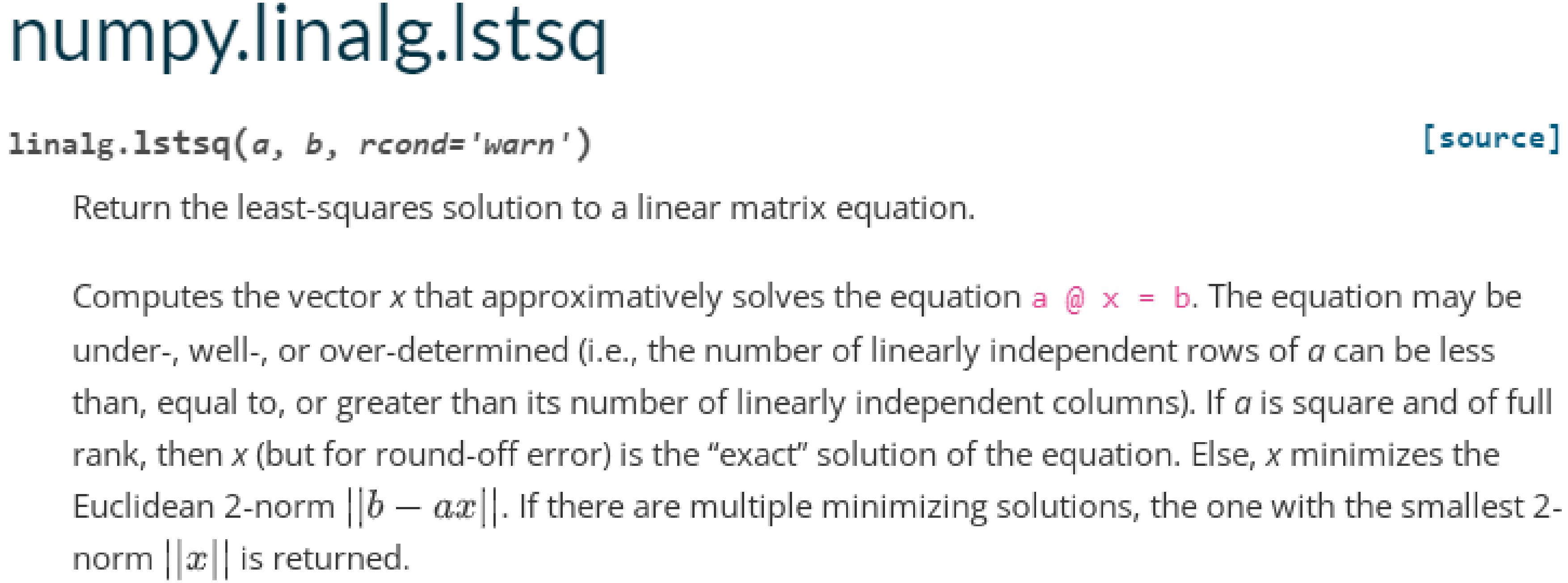
Line Fitting

- Given a point set with noise and outliers, estimate the parameters: $y = kx + b$
- Implement least-squares solution
- Implement RANSAC (300 iterations)

1. randomly choose a small subset from the noisy point set ;
2. compute the least-squares solution for this subset;
3. compute the number of inliers, if the number exceeds the current best result, update the estimation

Line Fitting

https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html



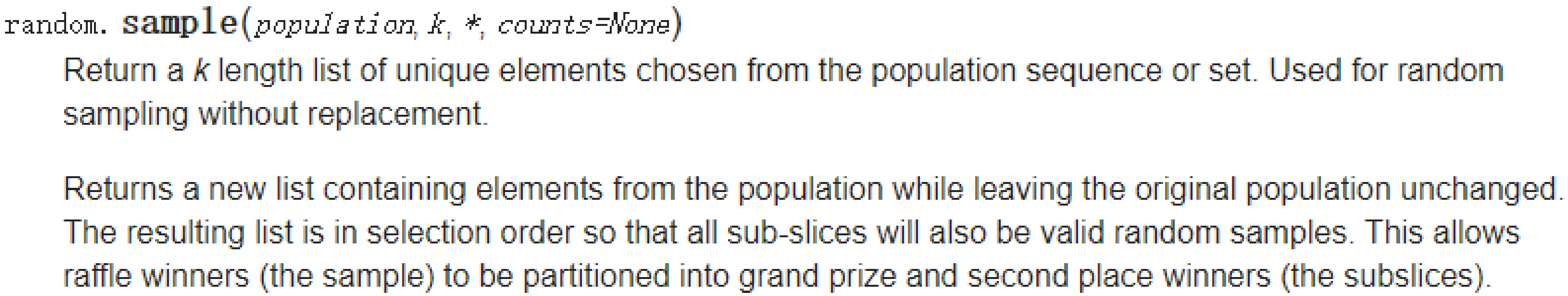
numpy.linalg.lstsq

`linalg.lstsq(a, b, rcond='warn')` [source]

Return the least-squares solution to a linear matrix equation.

Computes the vector *x* that approximatively solves the equation `a @ x = b`. The equation may be under-, well-, or over-determined (i.e., the number of linearly independent rows of *a* can be less than, equal to, or greater than its number of linearly independent columns). If *a* is square and of full rank, then *x* (but for round-off error) is the “exact” solution of the equation. Else, *x* minimizes the Euclidean 2-norm $||b - ax||$. If there are multiple minimizing solutions, the one with the smallest 2-norm $||x||$ is returned.

https://docs.python.org/3/library/random.html

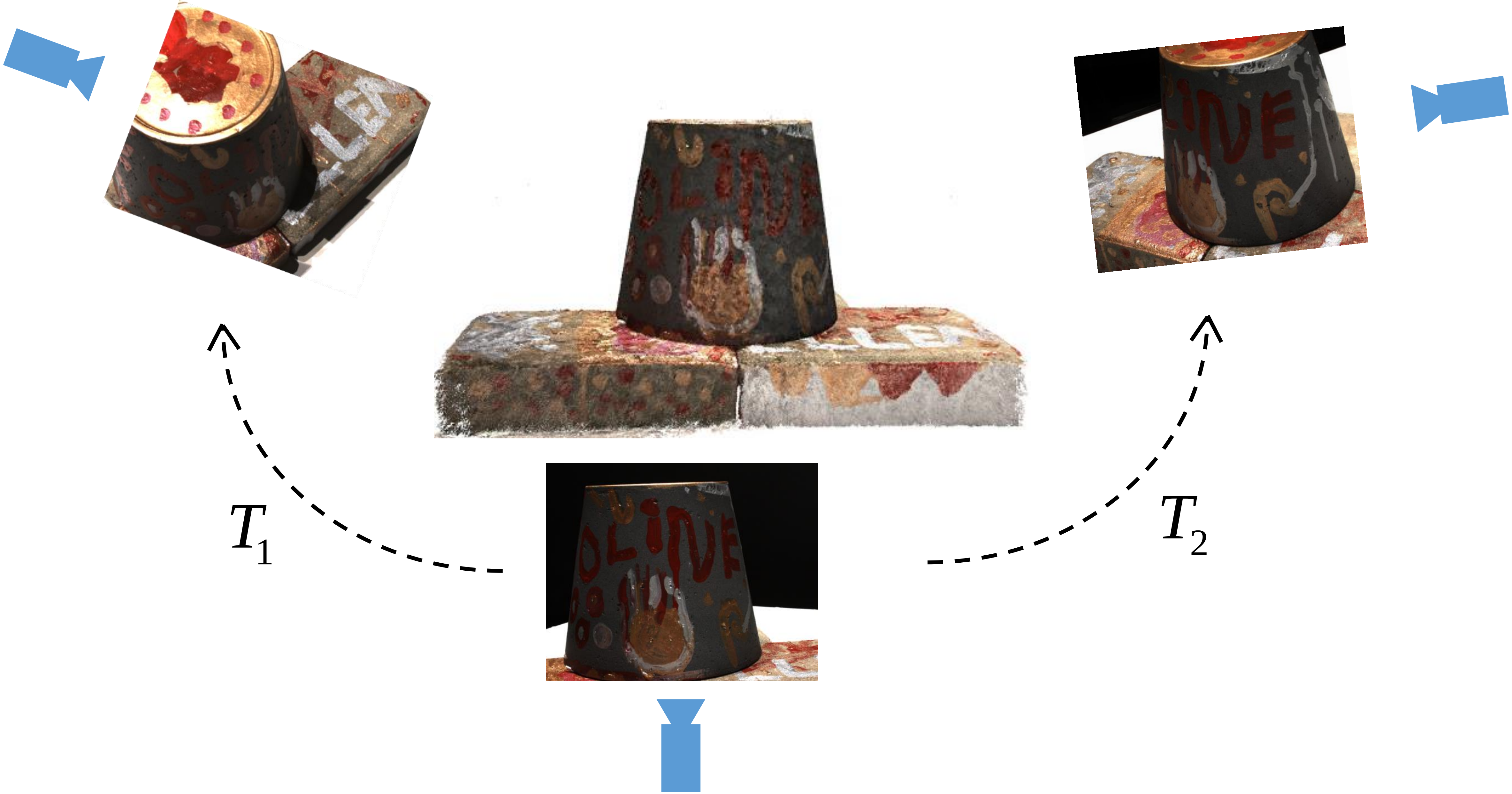


`random.sample(population, k, *, counts=None)`

Return a *k* length list of unique elements chosen from the population sequence or set. Used for random sampling without replacement.

Returns a new list containing elements from the population while leaving the original population unchanged. The resulting list is in selection order so that all sub-slices will also be valid random samples. This allows raffle winners (the sample) to be partitioned into grand prize and second place winners (the subslices).

Multi-View Stereo



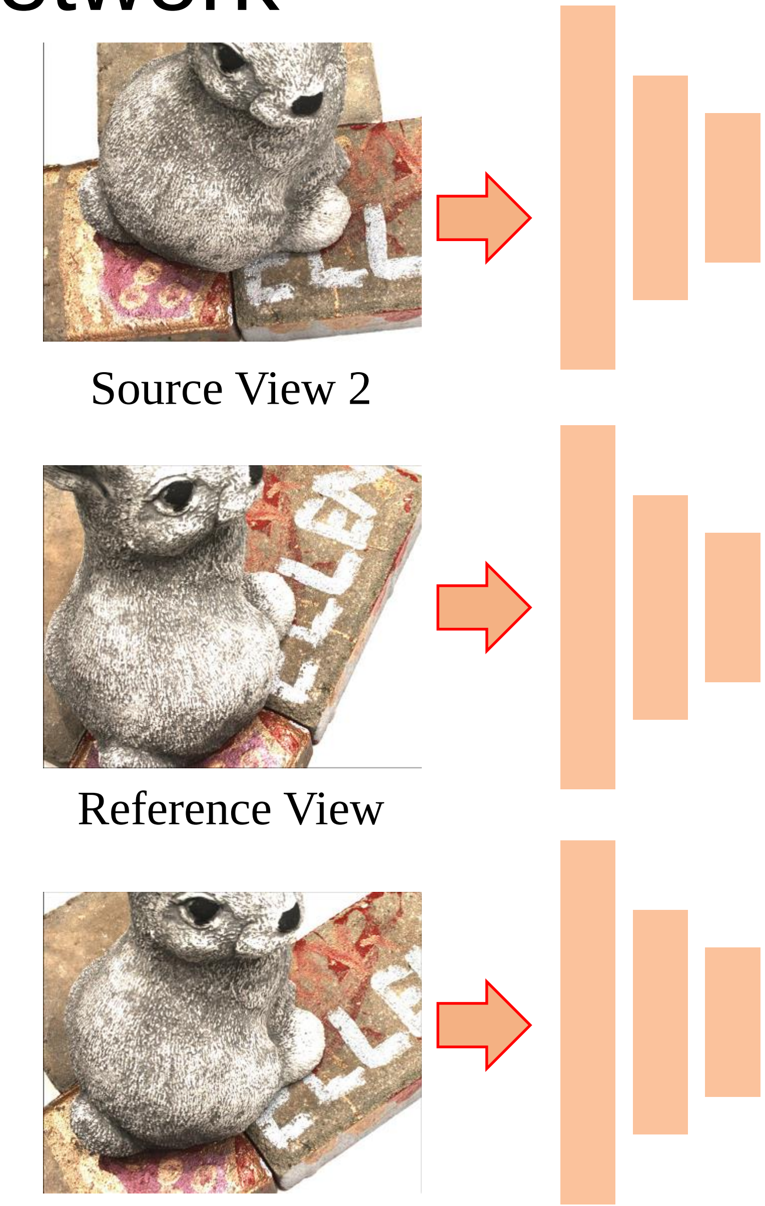
Network

Source View 2

Reference View

Source View 1

Network



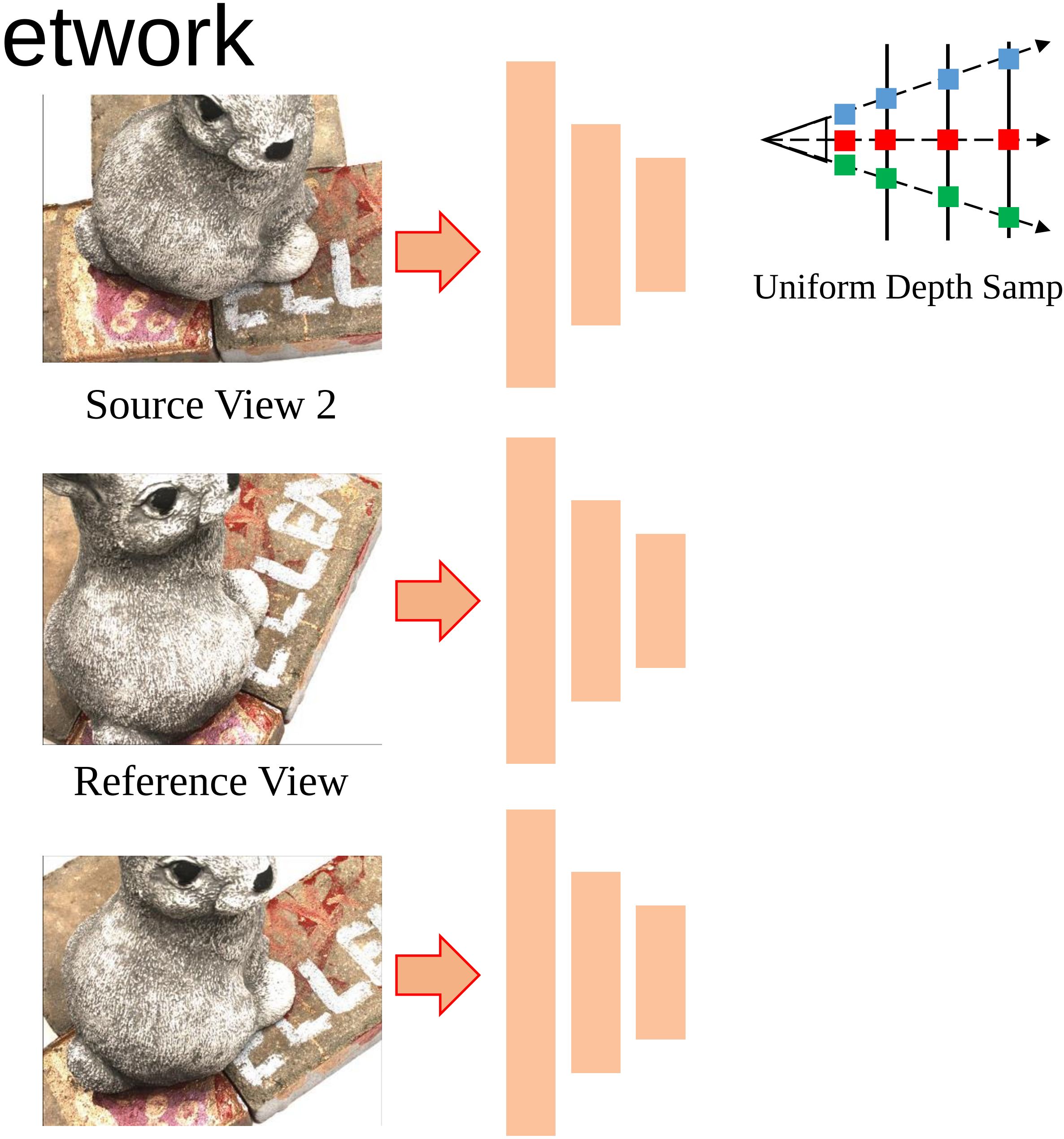
Source View 2

Reference View

Source View 1

Feature

Network



Uniform Depth Samples

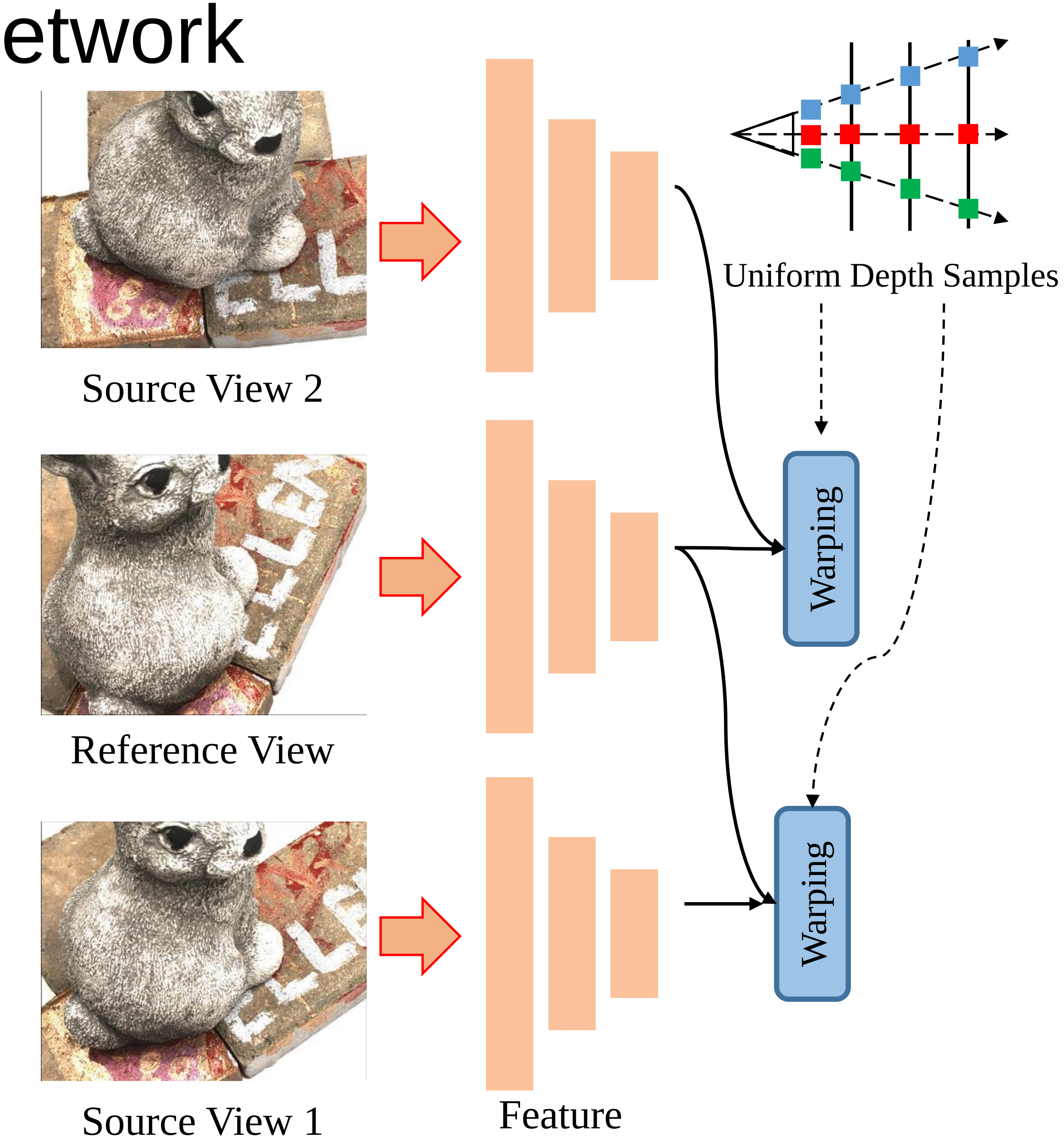
Source View 2

Reference View

Source View 1

Feature

Network



Uniform Depth Samples

Source View 2

Warping

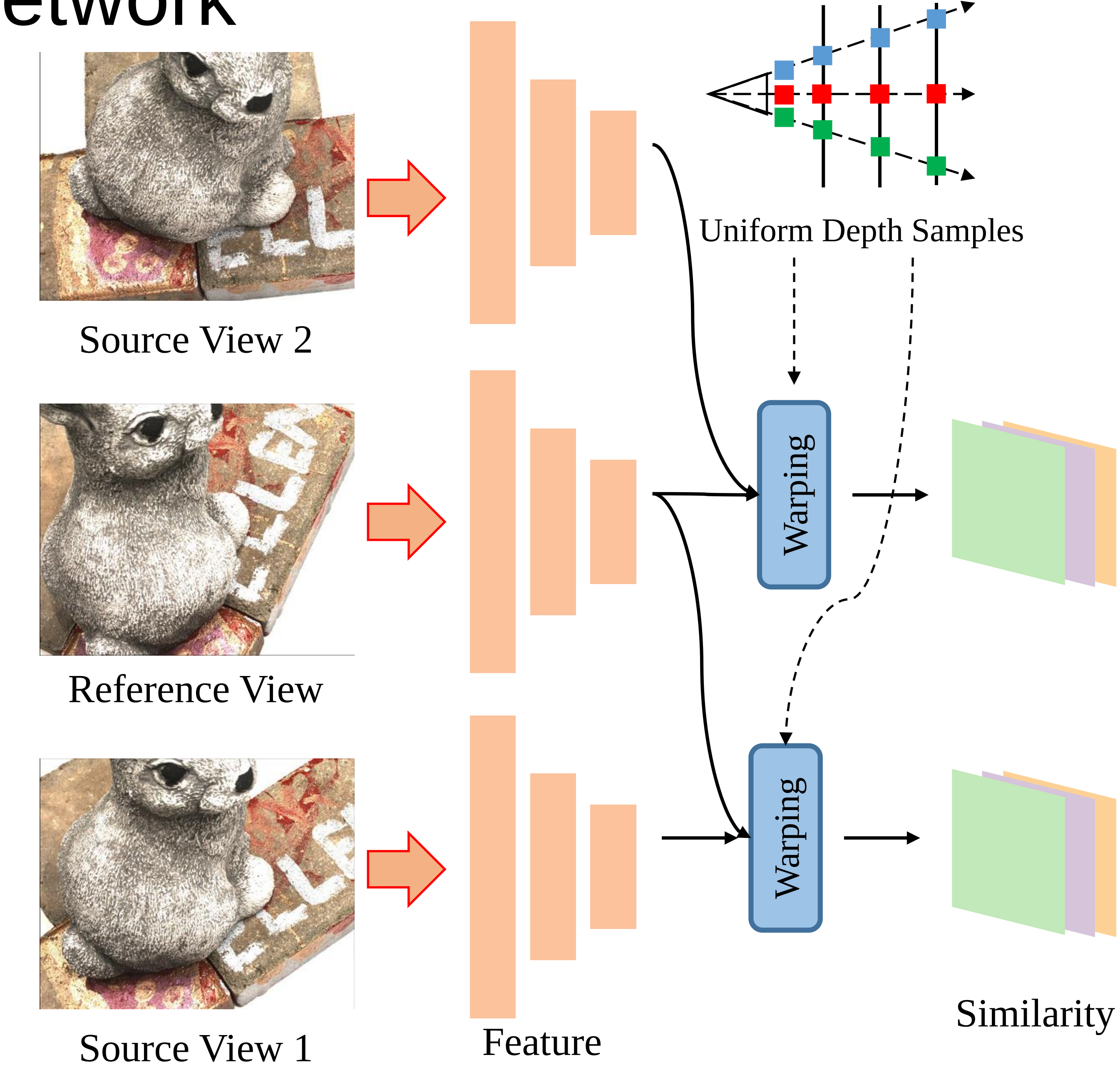
Reference View

Warping

Source View 1

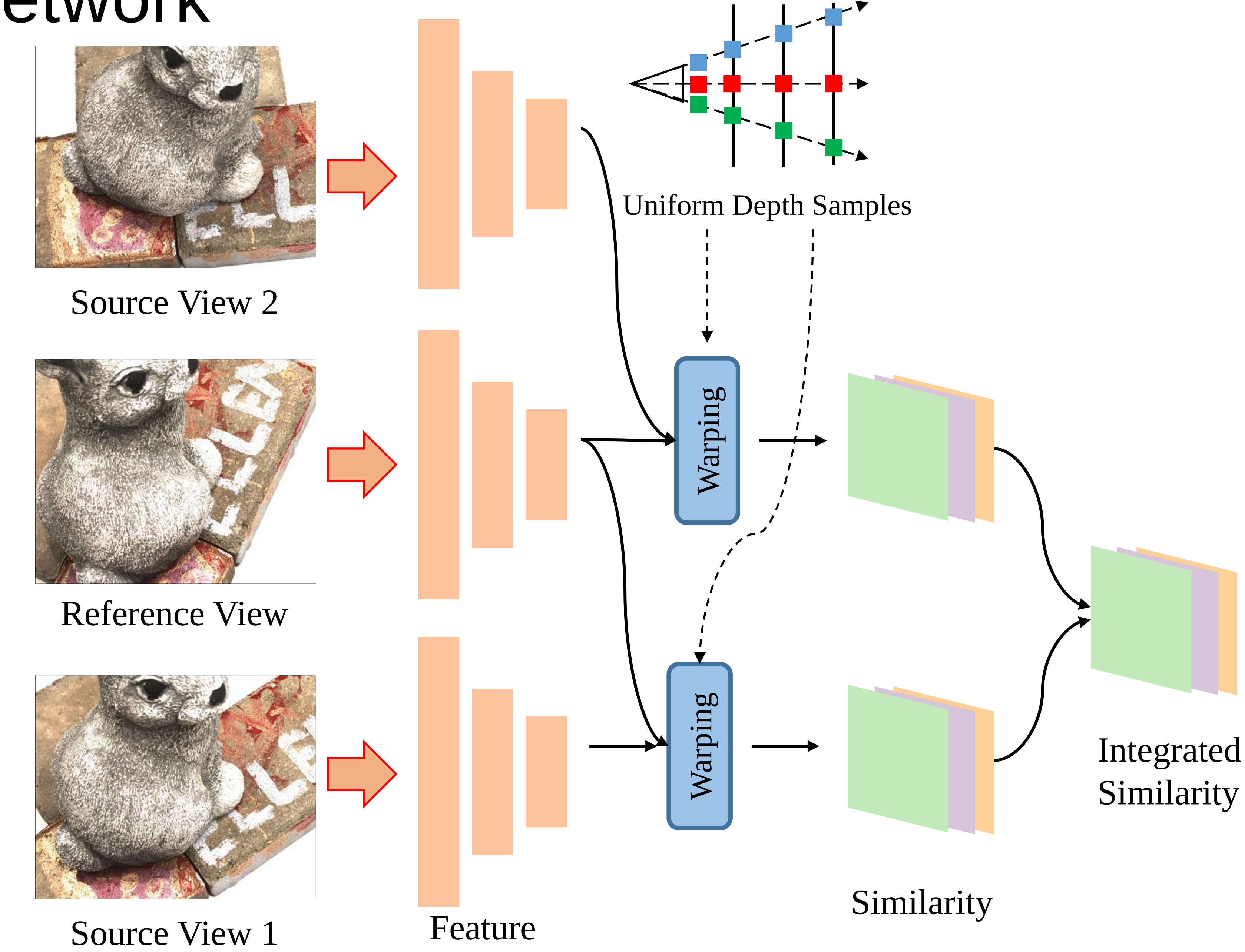
Feature

Network

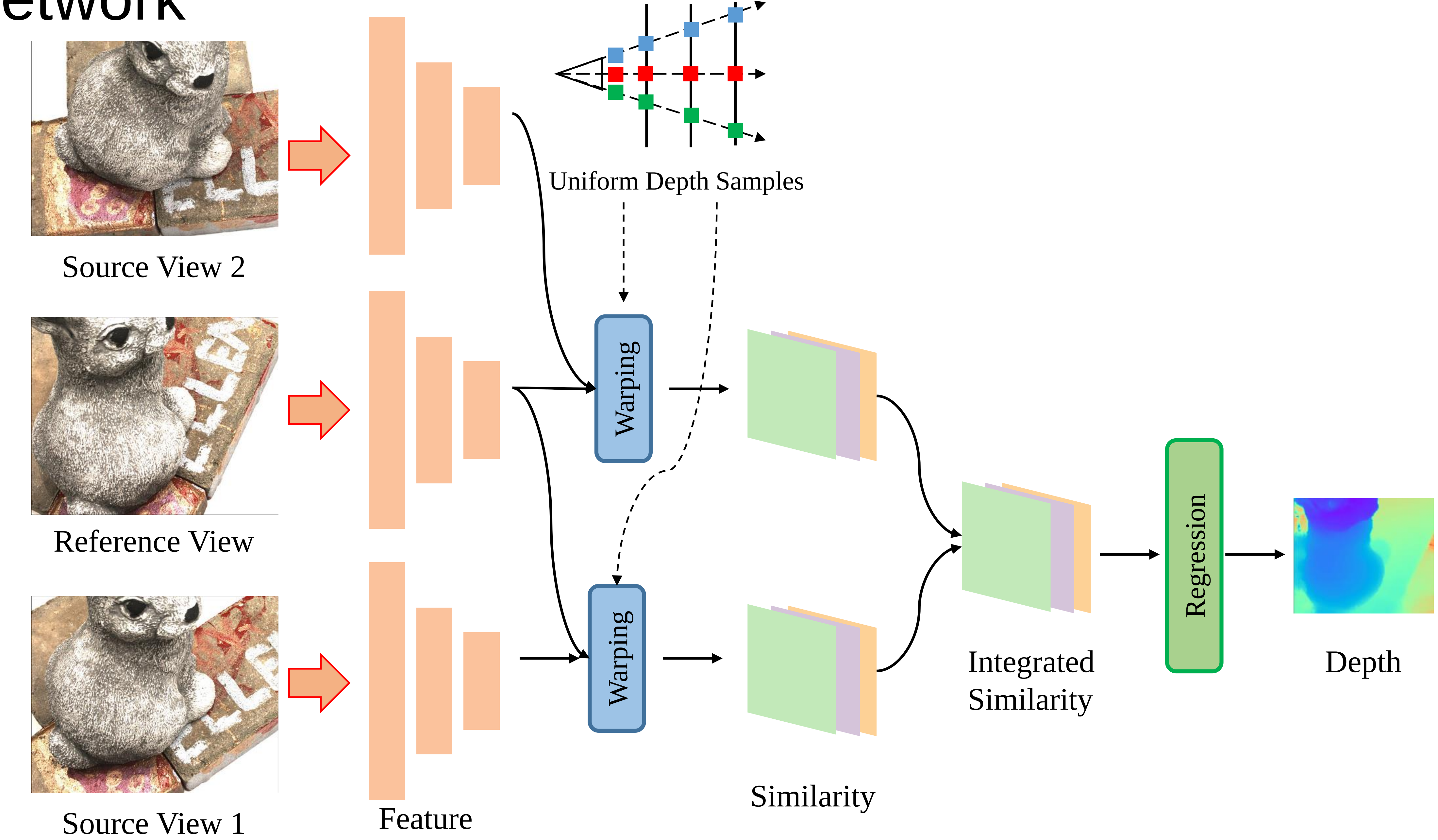


Uniform Depth Samples
Source View 2
Reference View
Source View 1
Feature
Warping
Warping
Similarity

Network

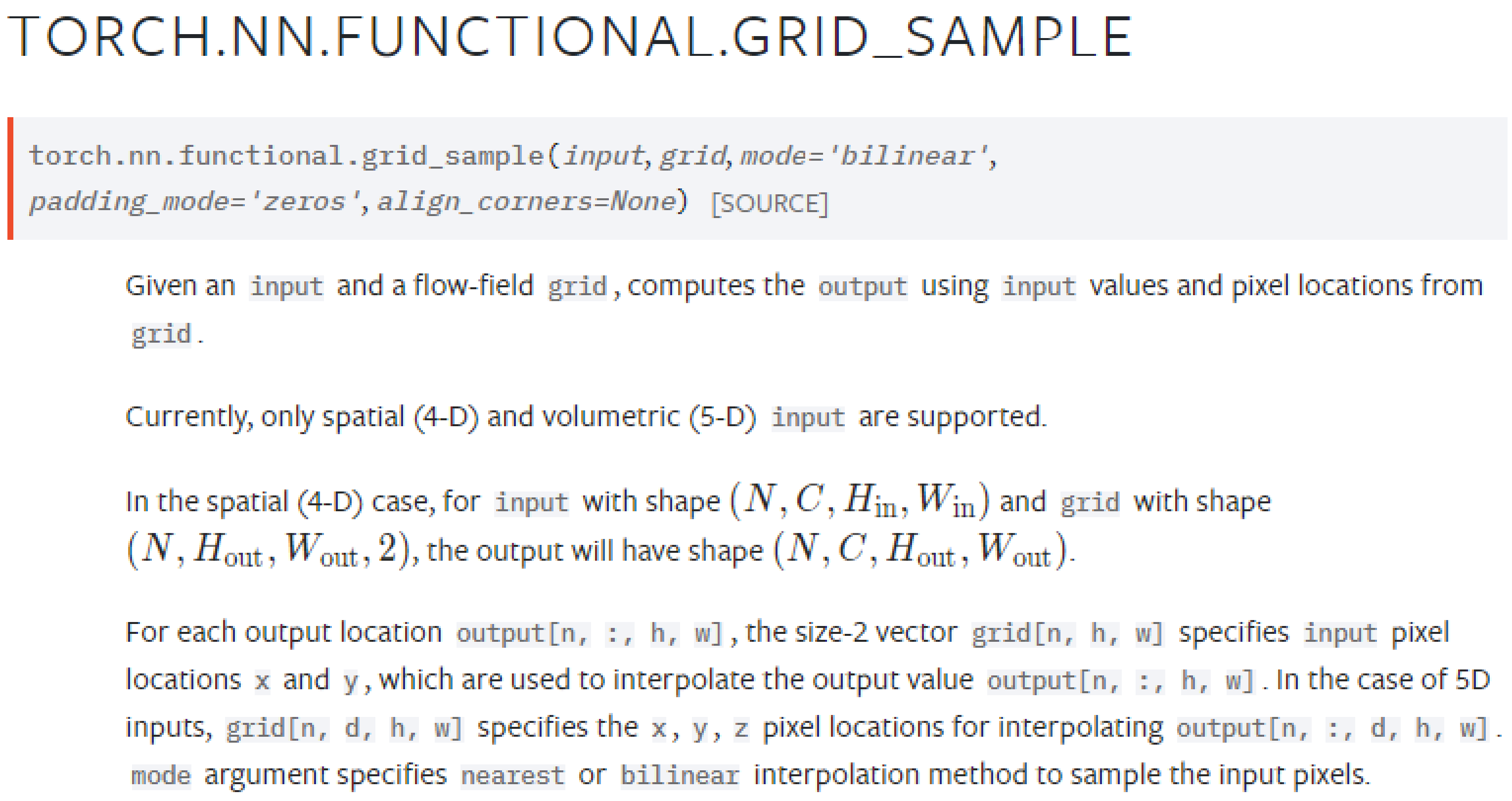

Uniform Depth Samples
Source View 2
Reference View
Source View 1
Feature
Warping
Warping
Similarity
Integrated
Similarity

Network


Uniform Depth Samples
Source View 2
Reference View
Source View 1
Feature
Warping
Warping
Similarity
Integrated Similarity
Regression
Depth

Warping

For pixel p in the reference feature and a depth value d_j , we find its corresponding pixel $p_{i,j} := p_i\left(d_j\right)$ in source view i , then we get the source feature of $p_{i,j}$



TORCH.NN.FUNCTIONAL.GRID_SAMPLE

```
torch.nn.functional.grid_sample(input, grid, mode='bilinear',
padding_mode='zeros', align_corners=None)
```

[SOURCE]

Given an `input` and a flow-field `grid`, computes the `output` using `input` values and pixel locations from `grid`.

Currently, only spatial (4-D) and volumetric (5-D) `input` are supported.

In the spatial (4-D) case, for `input` with shape $(N, C, H_{\text{in}}, W_{\text{in}})$ and `grid` with shape $(N, H_{\text{out}}, W_{\text{out}}, 2)$, the output will have shape $(N, C, H_{\text{out}}, W_{\text{out}})$.

For each output location `output[n, :, h, w]`, the size-2 vector `grid[n, h, w]` specifies `input` pixel locations `x` and `y`, which are used to interpolate the output value `output[n, :, h, w]`. In the case of 5D inputs, `grid[n, d, h, w]` specifies the `x`, `y`, `z` pixel locations for interpolating `output[n, :, d, h, w]`. `mode` argument specifies `nearest` or `bilinear` interpolation method to sample the input pixels.

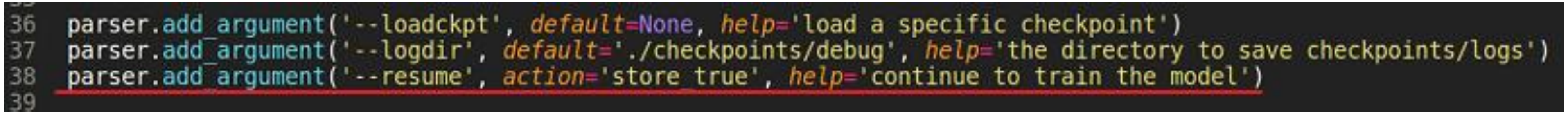
https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html

Dataset

- a part of DTU dataset, a large-scale indoor multi-view stereo dataset
- download link: https://polybox.ethz.ch/index.php/s/H7SWzSgIQJoSsR2
- Modify the path in **train.sh** and **eval.sh**

Training and Testing

- Training takes about 9 hours for 4 epochs (cpu-only), If the training process is stopped (e.g. trained 2 epochs), you can resume the training with argument ‘resume’ (details in train.py)

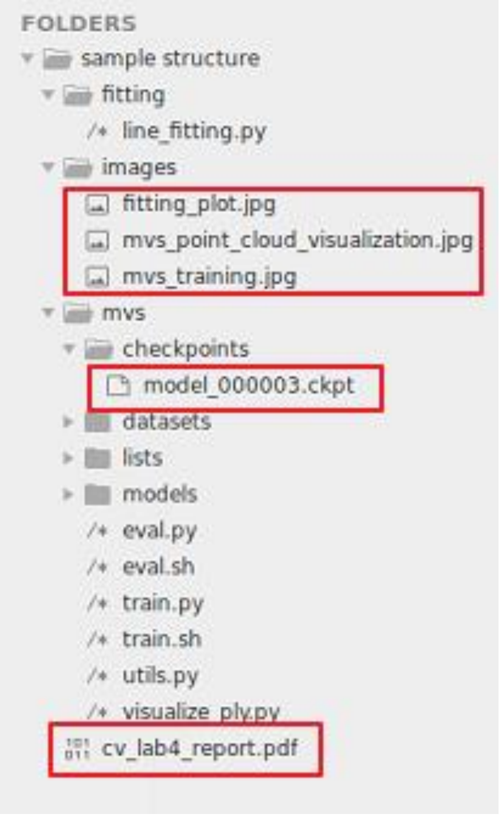


```
parser.add_argument('--loadckpt', default=None, help='load a specific checkpoint')
parser.add_argument('--logdir', default='./checkpoints/debug', help='the directory to save checkpoints/logs')
parser.add_argument('--resume', action='store_true', help='continue to train the model')
```

- With the trained model, do depth estimation and point cloud reconstruction (eval.py)

Summary

- Hand in report, images and codes as a zip to moodle before Friday, December 3, 23:59.
- For multi-view stereo, also upload the trained model: model_000003.ckpt



```
FOLDERS
▼ sample structure
  ▼ fitting
      line_fitting.py
  ▼ images
      fitting_plot.jpg
      mvs_point_cloud_visualization.jpg
      mvs_training.jpg
  ▼ mvs
    ▼ checkpoints
        model_000003.ckpt
    ▸ datasets
    ▸ lists
    ▸ models
      eval.py
      eval.sh
      train.py
      train.sh
      utils.py
      visualize_ply.py
    cv_lab4_report.pdf
```