Geometric computer vision v.2021.1: Homework assignment #1

Alexey Artemov

15 March 2021

1 Introduction

Your goal for this homework is to construct a regression estimate of a characteristic defined on a 3D point cloud of a 3D shape, using a set of range-images of the same 3D shape (2D projections) obtained using a virtual 3D scanner. This technique for obtaining the estimate is general and is often referred to as *multiple view consistency* in computer vision.

Relevant literature on these topics are present in [1, 2].

2 Definitions

Consider a set of points $\mathbf{X}^\ell = \{x_i\}_{i=1}^\ell$ in $\mathbb{R}^3$ and a set of *sharp feature curves* $\Gamma = \{\gamma_k\}_{k=1}^K$ in $\mathbb{R}^3$. We associate each point x_i with its closest point p_i belonging to one of the curves in Γ, such that $||p_i - x_i|| = \min\limits_{\gamma_k \in \Gamma} \inf\limits_{p \in \gamma_k} ||p - x_i||$, and define the truncated *distance-to-feature field* as $d^\varepsilon(x_i) = \min(||p_i - x_i||, \varepsilon)$, where ε is the truncation radius. Figures 1–2 demonstrate what the field $d^\varepsilon(x)$ looks like in practice, for a point cloud sampling of a real 3D shape.

In practice it is beneficial to use depth images of the 3D shapes to be able to use convolutional neural networks (CNNs) that are well-known to deliver state-of-the-art performance for prediction in per-pixel labeling tasks.

3 The task

We have trained for you an image-based neural network $\text{CNN}(\theta^*, I)$ that accepts an input depth image I and produces per-pixel distance-to-feature values $\widehat{d}$. However, these predictions at the same 3D locations differ from one view to another due to the variability in context captured in each particular view (see Figures 1–2).

Your goal is to develop a procedure to obtain multiple view-consistent predictions using a set of single-view predictions and known camera parameters for each view.

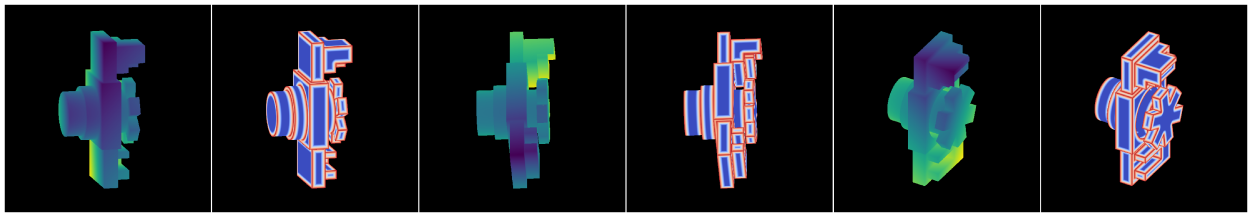

Figure 1: Ground-truth projection of a 3D shape and its accompanying distance-to-feature field onto a depth image. Note how the field is colored red (indicating closeness to feature) both at the contour and in the interior of the shape. The field is consistent across various locations in the 3D shape.

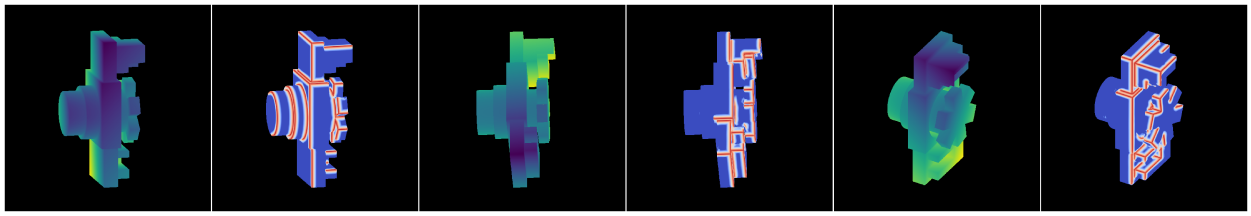

Figure 2: Predicted per-pixel distance-to-feature field for a 3D shape and its accompanying depth image. Note how the field is colored red (indicating closeness to feature) only in the interior of the shape. This is because the CNN model used to predict this field cannot tell the difference between a sharp contour and a smooth contour due to a lack of spatial context. The field is *not* consistent across various locations in the 3D shape.

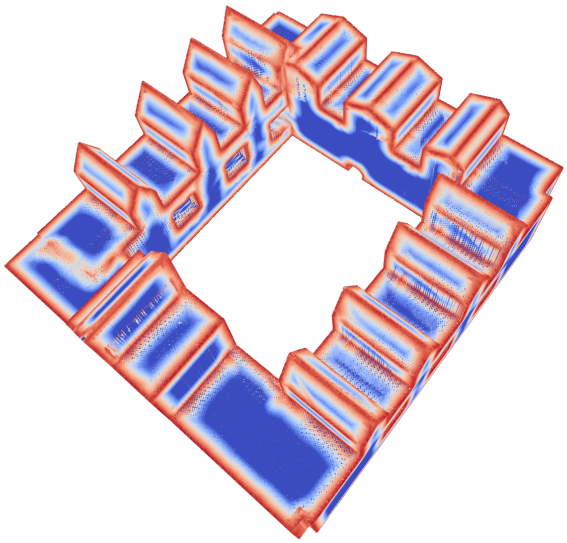

Figure 3: A 3D point cloud obtained using an implementation of the fusion algorithm described in Section 3.1.

- The input to your procedure is the list of depth images I_i, single-view predictions $\widehat{d}$, and camera parameters K_f, K_s (shared across views). These are present in Figure 2.
- The output of your procedure is a point cloud $P = \{p_k\} \subset \mathbb{R}^3$ obtained by merging reprojected depth images $\{I_i\}$, with per-point distance-to-feature values $D(p_k) \geqslant 0$. These are present in Figure 3.

3.1 Algorithm

The algorithm for interpolating predictions considers all pairs (i, j) of N views of the same 3D shape. Let I_i, I_j denote depth images (i.e., $I_{i,j}(u, v) = z$ is a depth value where (u, v) is a pixel) and $\widehat{d}_i, \widehat{d}_j$ denote the distance-to-feature fields predicted for views i and j, respectively.

Step 1. Using known camera parameters, *re-project* image I_j onto the view i, computing $\widetilde{I}_j$.

Step 2. For each pixel (u, v) in $\widetilde{I}_j$, find k nearest neighbours in view i: $\mathbb{P} = \{I_j(u', v'), (u', v') \in \mathrm{NN}_i(u, v)\}$.

Step 3. Compute an *interpolated* distance-to-feature value $\widetilde{d}^i_j(u, v)$ using the set $\mathbb{P}$ and distance-to-feature values for pixels in $\mathbb{P}$: $\{\widehat{d}_i(u', v'), (u', v') \in \mathrm{NN}_i(u, v)\}$

After this procedure is complete, for each original distance-to-feature value $\widehat{d}_i(x)$ predicted by the CNN we will obtain up to $N-1$ additional (interpolated) values $\widetilde{d}^1_i(x), \ldots, \widetilde{d}^{N-1}_i(x)$.

We then *aggregate* the predictions for the point cloud by taking a min over predicted and interpolated distance-to-feature values: $\widehat{d}^{\mathrm{AGG}}(x) = \min\{\widehat{d}_i(x), \widetilde{d}^1_i(x), \ldots, \widetilde{d}^{N-1}_i(x)\}$.

More implementation details are presented in the accompanying source code. Note that the description above is a high-level one and may lead to different implementations depending on how we define "interpolation", "reprojection" or "aggregation".

3.2 Problems

1. The multiple view interpolation approach described in code assumes *fusion in image space* (i.e., using real-valued UV coordinates on canvas). You must first complete the task of interpolating using this approach. To assess your performance, you can measure mean square error between your fused distance-to-feature field and the ground-truth field.

2. (Bonus task, 2 points) *More stable interpolation.* Note that bilinear interpolation method used in the code (specifically `scipy.interpolate.interp2d`) produces many warnings due to diagnosed instabilities when interpolating. Use `fitpack.bisplrep` and `fitpack.bisplev` instead to obtain more stable estimates.

3. (Bonus task, 2 points) *More elaborate interpolation.* Note that just taking a min is a very simple way of combining predictions that does not take into account any statistics observed in neighbouring points (or even in the same point). Design and implement a different (and more effective) combining strategy for aggregating predictions. Put the code for this strategy into `<repo>/gcv_v20211_hw1/fusion/combiners.py`.

4. (Bonus task, 6 points) *Fusion in pixel space.* Note that, when we compute the real-valued coordinates (u, v) for image $\widetilde{I}_j$, we do not form an actual pixel 2D image. (Instead, we are left with a point cloud living in the image plane). Perform an experiment, where an actual 2D pixel image is being formed. To this end, you must implement a visibility constraint: when assigning depth values to (rounded) pixels (k, m), check for the existence of another pixel mapping into the same pixel (k, m), and having smaller depth. We have not taken visibility constrains into account, meaning that re-projection from view i into view j may result in a depth image being inverted (points farther from the viewer appear in the pixel grid, while points closer do not) due to pixel. Implement the rest of interpolation procedure as previously.

4 Resources

4.1 Data

You are presented with a set of `.hdf5` containers with ground-truth images (for reference) and predicted images. The validation dataset is available for download by this link: validation.zip.

Additionally, when evaluating your solution, we will use a separate hidden test set (unavailable to students).

4.2 Code

You have the access to the framework that implements

- reading and saving datasets,
- visualizing point-cloud and range-image datasets,
- calling the fusion scripts and all the necessary wrappers,
- transforming between pixel and points, and transforming between points defined in the global frame and in the camera frame.

5 Submitting the solution

You must submit:

1. The *repository* with a set of complete scripts that may be used to perform fusion of distance-to-feature predictions.
2. *Predictions* that you were able to obtain using your code.
3. An accompanying *document* (in PDF format) describing your approach and solution.

The deadline for the submission of the homework is **Sunday, March 28, 2021.** We will release the final grades for the homework by **Sunday, April 4, 2021.**

The evaluation procedure by the TAs will be to:

1. (6 points) Run your submitted scripts on all of the validation and test images, obtaining multiple view consistent predictions. We will use the initially provided script `<repo>/scripts/fuse_images.py` to run your code, so please do not change its invocation interface.
2. (2 points) Examine the implementation that you provided for correctness. This is done by reading your implementation code in the file `<repo>/gcv_v20211_hw1/fusion/interpolators.py`, no other files will be checked (except for the bonus problems).
3. (2 points) Examine your attached document for a description of your code. The description must include a detailed comment (in English language, not code) for each of the TODOs found in code that you had to implement.
4. (10 bonus points) Examine the bonus tasks described in Section 3.2.

References

[1] Albert Matveev, Alexey Artemov, Ruslan Rakhimov, Gleb Bobrovskikh, Daniele Panozzo, Denis Zorin, and Evgeny Burnaev. Def: Deep estimation of sharp geometric features in 3d shapes. *arXiv preprint arXiv:2011.15081*, 2020.

[2] Tejas Khot, Shubham Agrawal, Shubham Tulsiani, Christoph Mertz, Simon Lucey, and Martial Hebert. Learning unsupervised multi-view stereopsis via robust photometric consistency. *arXiv preprint arXiv:1905.02706*, 2019.