Using image registration for computational dolly zoom

Arjun Dhawan (akdhawan) and Gleb Shevchuk (glebs)

***Abstract*—In this project, we investigate different methods for performing computational dolly zoom using image registration. We define computational zoom as a four step pipeline consisting of feature selection, feature matching, transformation, and interpolation. To establish the most effective method for performing computational dolly zoom on static images, we examine several options for each step and determine the combination of techniques that creates the best dolly zoom. Finally, to test theses approaches, we apply the approaches on a variety of static images taken at different depths and analyze the differences between qualitative and quantitative results on each combination of methods. Our code is located at https://www.github.com/glebshevchukk/231a_zoom**

I. Introduction

We describe a system that, given N pictures captured in a straight or angled line, outputs a video interpolating between the frames while controlling parallax between the foreground and background. This idea is commonly used in-camera with effects like the dolly zoom, which moves a camera backward on a dolly while zooming in on the foreground. This gives the appearance of a character or object "falling out" of the background. By doing this out-of-camera, we hope to give additional control and allow anyone to create rich, programmable parallax between foreground and background by simply taking a few pictures. Finally, we hope to make this approach applicable to a number of different scaling and transformation effects by creating a general framework to identify features in a scene across images, segment out foreground from background, and enforce constraints on the resulting video. In computer vision, this technique is formally known as image registration because it involves fitting images to each other using common features.

II. Problem Statement

More formally, we can define our problem as follows: given a source and target image, a parallax method transforms the target into the source such that a subset of matching features (in the foreground, background, or other important region) map to the same points in image space. By changing the selection of points, we re-create camera effects like the dolly zoom and reverse dolly zoom. We do so using traditional 3D reconstruction techniques and test our approach using image sequences collected with smartphone cameras. Finally, we evaluate our approaches quantitatively by measuring the distance between matched features in the final transformations. For data, we collected around 10 sequences with an average of 5 frames per sequence.

III. Related Works

In order to create programmable parallax, we require four distinct steps: feature detection, foreground segmentation, image transformation, and interpolation.

Feature detection is perhaps the richest of these three segments and has seen critical advances over the course of computer vision's history. [13] provides a comprehensive survey of this space, discussing the evolution of local feature detectors and trade-offs of approaches like Harris corner, DoG, SURF and other featurizers. For this project, we use ORB features because they have been shown to work well for tasks like SLAM over the last ten years and their approach is supported by Python libraries like OpenCV.

Next, foreground segmentation is often tied to and improved by advances in scene segmentation. Recently, deep-learning techniques like Mask-R-CNN [7] have shown incredible performance on such segmentation tasks. However, for our use case, because we will have access to the true depth maps from using smartphone cameras, we plan to sidestep this segmentation problem by either being given known depth masks from an iPhone or by manually performing segmentation using the Watershed algorithm [4].

Next, a number of recent works have proposed to solve the novel view synthesis problem by performing nonlinear warping[2] or directly predicting pixel flow and depth or work[6] [11], usually using function approximators like neural networks. Many of these approaches, however, do not explicitly focus on in-frame transformations and instead concern themselves with longer-term sequence prediction. This, compounded with unknown poses and a focus on dynamic scenes, makes these approaches significantly more tricky than the one we hope to explore. In practice, because we deal with fairly constricted image sequences, we again sidestep the more difficult view synthesis problem and instead choose to parameterize our transforms as more traditional affine matrices, homography matrices, and splines [1]. We will go over these more in depth in the methods section.

Finally, there has also been a rich body of work in interpolating frames between key frames. Though linear interpolation has been used most regularly, a number of other more traditional approaches have used adaptive convolutions [12], phase-based motion [10], and flow estimation [8].

More recently, a number of neural-network based approaches have been proposed to solve the interpolation problem, including [9], which trains a Generative Adversarial Network using cyclic consistency loss to produce intermediate

frames, [14], which utilizes a recurrent pyramid network structure, and [3] which uses known depth information to aid in optical flow estimation.

IV. Methods

Accomplishing this task requires four key components: performing foreground segmentation, doing feature detection and matching, applying image transformation, and finally interpolating between frames.

Each of these can be done in a number of ways and for the sake of experimentation, we examine two ways of doing foreground segmentation, one way of doing feature matching, three ways of performing a transformation, and two ways of performing interpolation. These approaches are summarized in the below table:

segment	feature-match	transform	interpolate
given	ORB+BF	affine	linear
manual		homography	flow
		thin-plate spline	

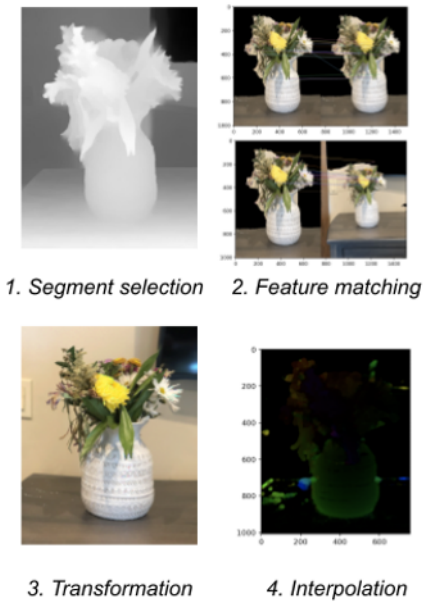

Fig. 1. Four core steps to performing image registration.

A. Foreground segmentation

We examined two different methods for foreground segmentation. The first method was using portrait mode from an iPhone camera and obtaining the depth results from the pre-segmented image. The second was manually segmenting the image sequences using the watershed algorithm.

Though estimating the foreground of an image correctly is still an open problem, especially in the monocular setting, we simplify it by using additional depth information made available through a smartphone camera. Namely, by using Portrait Mode on the iPhone camera, we are able to capture both RGB and depth information. Then, by thresholding the depth at a certain value, we are able to extract all foreground elements in an image. Therefore, the first step in our pipeline involves using Portrait Mode to capture N images of a subject, using the Exiftool software to extract depth information, then thresholding each image and seperating it into foreground and background elements.

Fig. 2. RGB and accompanying depth map captured using Portrait mode.

An alternate method for segmenting the foreground is a form of manual segmentation. Since manually indicating pixel by pixel what is the foreground is inefficient, we use the watershed algorithm to speed up the process. The implementation of the watershed algorithm in this project presents the user with an image. From this point, they use two different marker colors to indicate significant points on the image delinating the foreground from the background. The exact specifics of the watershed algorithm are beyond the scope of this paper, however, the watershed algorithm utlizes these user defined marks to then flood pixels surround the marks and define different portions of an image based on marker color. The name watershed comes from the fact that the algorithm treats the 2d image as a topological map with peaks and valleys and the user defined marks act as basins. From this point, flooded pixels flow towards the user defined marks thus making boundaries for segmentation to be used further in the pipeline. An example of the use of the watershed algorithm to segment the foregroudn from the background is shown below.

Fig. 3. User marked RGB and accompanying depth map created using implemented watershed algorithm.

B. Feature detection

Next, we have to identify features in all images and match them. In dolly zoom, we only select features from

the foreground while in reverse dolly zoom, we only select features from the background of each image. Using OpenCV, we first identify ORB (Oriented FAST and Rotated BRIEF) features. Next, we use a brute force matcher that minimizes the squared distance between a source feature descriptor and all target descriptors. After this step, we establish feature correspondences between each target image (N-1 in total) and the original source image (the 0th image). An example of this fearure mapping can be seen below.

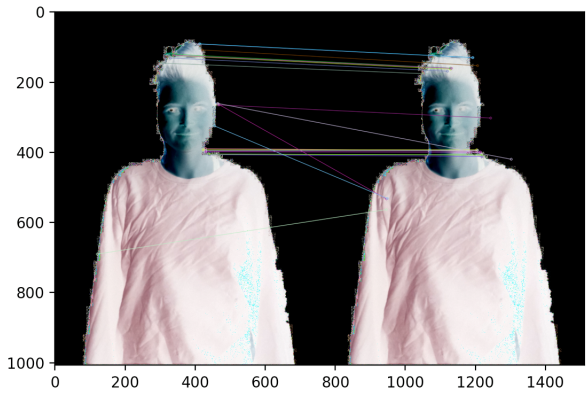

Fig. 4. Matched features between sequential segmented images using ORB features

C. Feature transformation

To simplify the feature transformation step, we assume that all images are strictly sorted by distance or rotation. We can then parametarize the transformation between each pair of images using estimated affine matrices, homography matrices, and splines.

For the first two methods, we estimate the transform as either an affine or projective matrix, where the only distinction is in number of controllable parameters, then perform a affine or projective transform.

Formally, we find the affine transform between each frame by initializing a random affine matrix with 4 controllable parameters:

$$A = \begin{bmatrix} s * \cos(\theta) & -s * \sin(\theta) & x \\ s * \sin(\theta) & s * \cos(\theta) & y \end{bmatrix}$$

Then, using RANSAC, we optimize towards the optimal affine matrix that minimize the distance between the target matched points from the original image denoted and the transformed source points $Ax_s \forall s \in S$.

We perform the same process for the latter two methods but with different parameterized representations, where the homography matrix expressed as:

$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

with the only constraint being that the final found matrix is normalized such that the final entry i is 1, and the thin-plate spline [5] being represented as a per-point displacement defined by the equation:

$$d(x, y) = a_1 + a_2 x + a_3 y + \sum_{i=1}^{n} w_i U(P_i - (x, y)) \quad (1)$$

where a_1, a_2, a_3, w_i are controllable parameters for each source point and U is a distance function.

D. Interpolation

Finally, after transforming each target image, we interpolate between these "key frames" in order to produce a final, smooth parallax video. We can do this two ways: by either linearly interpolating between or by using an estimated optical flow. For the former, we simply specify a number of intermediate frames then produce an interpolated video by taking a weighted sum of previous and next key frames. For the latter, we perform optical flow estimation using the Lucas-Kanade method then interpolate each pixel by using the found flow.

V. Results

In order to effectively determine what composes a good computational zoom, we did numerous tests between static image sequences to determine the best pipeline.

The segmentation appears to perform better on people rather than objects. Additionally, it appears to perform better on situations where the subject is more distinct from the background. Both of these insights are fairly intuitive, however give us direction on future implementations of foreground segmentation In addition to the implementation using the Exifol tool, a manual segmentation method using the watershed algorithm was used to segment the foreground from the background. In future iteration of this pipeline, an edge detector method such as canny edge detection could be used in combination with the portrait mode to give us a better estimates of depth. In all casesthough, segmentation provided good results that were usable by the algorithm. As a result any of the two methods, manual segmentation or segmentation using the Exiftool could be used in the pipeline.

For the interpolation type, we explored both flow-based interpolation as well as linear interpolation. With a flow based interpolation, we saw a smoother interpolation of the foreground, but also saw more artifacts in the background. Contrary to the effect seen by a flow based interpolation, the linear method appears to have a more jittery foreground flow and a smoother background flow.

For the transformation portion of the pipeline, we explored 3 different methods using affine, homographic, and thin-plate spline transformation. These results are shown in Figure 8. From these, we see that the thin-spline implementation appeared to spiral the foreground and not produce the desired dolly zoom effect. Meanwhile, the simplest transformation, the affine transformation, produced the best results where the foreground appeared to be static while the background

was transformed. The was counter intuitive since it was the simplest to implement, however makes sense retrospectively since the affine transformation allows us to create perspective distortion which typically results in the subject retaining its features throughout the transformation. Furthermore, given that each frame was produced by moving the camera along a single axis, an affine matrix seems precise enough to capture that motion.

It is interesting, too, to contrast the qualitative results shown in Figure 8 against the quantitative results seen in Figure 7. To produce this second figure, we measured the average pixel difference between the first frame in each sequence and each subsequent frame that we were transforming using the different methods. Importantly, we only measured this difference over the foreground element for which features were being calculated. Ideally, the loss for each subsequent frame would be 0, as the used transform would map the foreground to the target frame foreground correctly. At the same time, we should see these results match with the visual accuracy of the qualitative ones.

Surprisingly, this does not seem to be the case. Across all transform strategies, we should see decreased performance as frames get farther away from the original. This makes sense intuitively; as a frame gets farther from the original, it should be harder to find matching features between it and the source frame. In turn, the average loss across all sequences should generally have an upward slope. However, this is not the case for 3 of the 4 sequences we tested with. In addition, we expected these results to match with the visual ones but found little correlation. For 2 of the 4 sequences, the spline-based transform appears to have the best performance, even though we know it creates severe distortions and produces the least visually compelling dolly zooms. Furthermore, results showing the difference between linear and flow-based interpolation show the flow-based approach having better performance. However, through visual inspection, we know that the flow-based approach produces stronger distortions than the linear one and leads to more unnatural zooms.

Finally, our current approach leads to some jitter in the final output. This is difficult to illustrate in still images, however, the progression of the dolly zoom effect formed from the interpolation methods and transformation methods discussed above is displayed below.

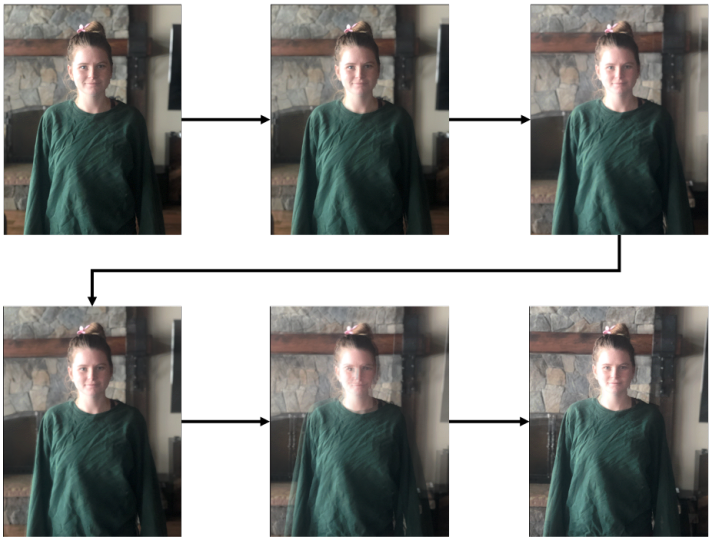

Fig. 5. Dolly zoom output estimated using affine transformation.

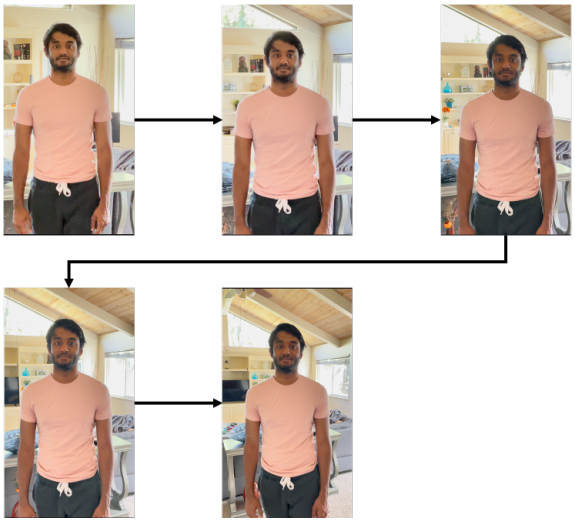

Fig. 6. Actual Dolly Zoom Effect output.

VI. Conclusion

In conclusion, the described pipeline allows us to produce dolly-zoom effects during post-processing. By implementing different approaches for each part of the pipeline, we explored what it takes to create a visually-consistent dolly zoom. In doing so, we found that the simplest approaches, using affine transformation and linear interpolation, produced the most stable results.

References

[1] Anubhav Agarwal, CV Jawahar, and PJ Narayanan. A survey of planar homography estimation techniques. *Centre for Visual Information Technology, Tech. Rep. IIIT/TR/2005/12*, 2005.

[2] Shai Avidan and Amnon Shashua. Novel view synthesis in tensor space. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1034–1040. IEEE, 1997.

[3] Wenbo Bao, Wei-Sheng Lai, Chao Ma, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. Depth-aware video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3703–3712, 2019.

[4] Lamia Jaafar Belaid and Walid Mourou. Image segmentation: a watershed transformation algorithm. *Image Analysis & Stereology*, 28(2):93–102, 2009.

[5] Fred L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions on pattern analysis and machine intelligence*, 11

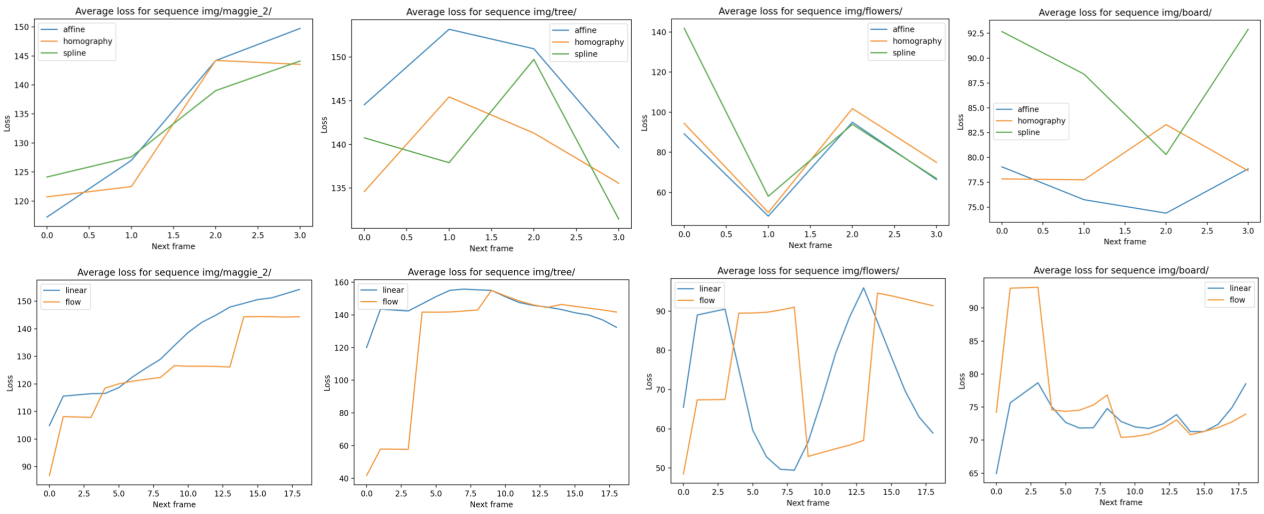

Fig. 7. Average pixel distance between the first target image and each transformed image. In the first row, we show this for different transform strategies across 4 image sequences. In the second row, we show this for 2 interpolation strategies.

(6):567–585, 1989.

[6] Ismael Daribo and Béatrice Pesquet-Popescu. Depth-aided image inpainting for novel view synthesis. In *2010 IEEE International workshop on multimedia signal processing*, pages 167–170. IEEE, 2010.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[8] Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9000–9008, 2018.

[9] Yu-Lun Liu, Yi-Tung Liao, Yen-Yu Lin, and Yung-Yu Chuang. Deep video frame interpolation using cyclic frame generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8794–8802, 2019.

[10] Simone Meyer, Oliver Wang, Henning Zimmer, Max Grosse, and Alexander Sorkine-Hornung. Phase-based frame interpolation for video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1410–1418, 2015.

[11] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020.

[12] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 670–679, 2017.

[13] Tinne Tuytelaars and Krystian Mikolajczyk. *Local invariant feature detectors: a survey*. Now Publishers Inc, 2008.

[14] Haoxian Zhang, Yang Zhao, and Ronggang Wang. A flexible recurrent residual pyramid network for video frame interpolation. In *European Conference on Computer Vision*, pages 474–491. Springer, 2020.

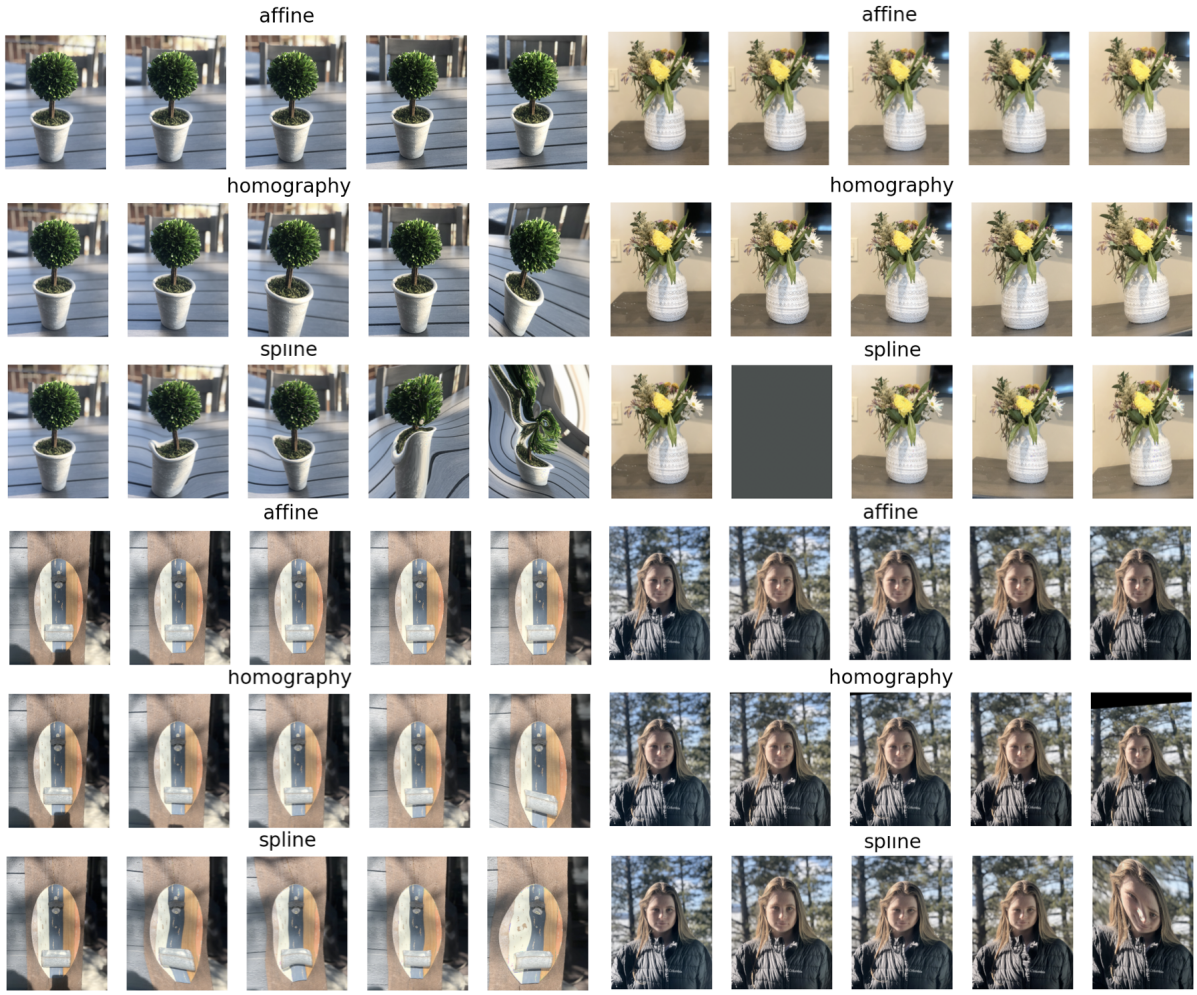

Fig. 8. Qualitative results from estimating different transforms.