Viewpoint Interpolation of Light Fields Using Variational Autoencoders

Valentin Wüst, Sebastian Gruber and Michael Dorkenwald
Heidelberg University
{v.wuest,s.gruber,m.dorkenwald}@stud.uni-heidelberg.de

Abstract

We present a fully convolutional variational autoencoder for light fields, which encodes stacks of aligned images through a deep network consisting of residual layers. The light field is thus reduced to a low-dimensional, highly disentangled representation. We utilize this representation to synthesize further viewpoints by linearly combining existing ones and then decoding the artificially created intermediate representation. Our approach uses unsupervised training with the aim to achieve good reconstruction as well as a semantically valuable latent space. We provide an extensive evaluation and show that the latent representation can be used to achieve almost arbitrary viewpoints within the boundaries of the provided data. This work was done as a project for the lecture 3D computer vision. Code is available at: `https://github.com/mdork/Light-Field-View-Synthesis-3DCV-Proejct`.

1. Introduction

Light fields can be obtained using an array of cameras located at different positions, with each taking a 2D image. They give a rich representation of a real world setting and enable a range of exciting applications such as post-capture refocusing and augmented reality [12]. From a physical standpoint, the 4D representation obtained is able to capture a large share of the light rays in a 3D volume, thus providing significantly more information than a normal 2D image. The advantages of dense light fields have been shown for various computer vision tasks, such as depth estimation [7], image-based rendering [11], and object segmentation [15]. Because acquiring light fields is expensive, the field of viewpoint synthesis has been an active research topic and can prove useful to support the mentioned applications.
Light field enhancement has been explored through various approaches, such as angular super-resolution, light field reconstruction, and view interpolation or synthesis. Our work tries to tackle this issue by training a variational autoencoder that aims to reconstruct aligned rows of light field images after passing them through a bottleneck. By also minimizing the Kullback-Leibler divergence, the network is forced to find highly semantic representations and thus disentangles the latent space. We use this to our advantage to be able to interpolate within the latent space by combining different viewpoint representations in a linear fashion in order to create a new one. As during training no special attention is directed towards this task, our method shows a high level of generalization.
We compare our generated viewpoints to those present in the data set and evaluate discrepancies qualitatively and quantitatively. As our architecture is lightweight and easy to use, it could potentially be used to reconstruct all desired intermediate viewpoints from relatively few original viewpoints, thereby reducing the cost of obtaining dense light fields.

2. Related Work

There are many deep learning based approaches to light field synthesis. Kalantari et al. proposed a two step CNN which first estimates the disparity from four corner views of a light field and then synthesizes new viewpoints based on the input images and the disparity [8]. A similar approach was pursued by Chaurasia et al., where instead of predicting the disparity the depth is estimated to then warp the images accordingly [3]. Zhou et al. use a network to predict a multi plane image representation based on a stereo image pair, which is then used to re-project the multi plane images to the desired viewpoint [14]. Chen et al. proposed to use cycle consistency to pre-train on videos and then generate new viewpoints, making use of the bidirectional mapping forcing the created viewpoints to match the input image [5].

3. Method

The task of interpolation between different light field viewpoints requires a generative network. These networks are powerful frameworks to synthesize novel samples. Moreover, we require a model which can learn an intermediate representation π of the input image $\boldsymbol{x}$. For those reasons, autoencoders are a natural choice and are used for

our project.

3.1. Variational Autoencoders

Variational Autoencoders (VAEs) [9, 10] approximate the true data distribution $p(\boldsymbol{x}, \pi)$, which is assumed to follow the generative process $p(\boldsymbol{x}, \pi) = p(\boldsymbol{x}|\pi)p(\pi)$. To optimize the intractable marginal log-likelihood a lower bound

$$L(p_\theta, q_\phi) := \mathbb{E}_{q_\phi(\pi|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\pi)\right] - D_{\mathrm{KL}}(q_\phi(\pi|\boldsymbol{x})||p(\pi)) \tag{1}$$

is introduced with neural networks for the decoder p_θ, and the encoder q_ϕ. $p(\pi)$ represents the prior on the latent space and is set to a standard Gaussian. The first term can be seen to optimize the synthesis quality of the decoder while the second part regularizes the encoder to match the prior distribution and by that, leads to a dense representation space which is needed for interpolation.
However, the trade-off between synthesis and regularization in optimizing Eq. 2 leads to a reduction of generation quality [4, 13]. To balance this trade-off, [2] introduced a β factor to weight the regularization term against the reconstruction term in Eq. 2. However, choosing an appropriate β factor is complicated. To alleviate this problem, we allow the network to have a certain information budget I_{max} which is easier to adjust. By that, Eq. 2 can be rewritten as

$$L(p_\theta, q_\phi) := \mathbb{E}_{q_\phi(\pi|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\pi)\right] - \rho(0, D_{\mathrm{KL}}(q_\phi(\pi|\boldsymbol{x})||p(\pi)) - I_{max}) \tag{2}$$

with ρ the ReLU activation function. Allowing a certain information budget I_{max} leads to a miss match between the Gaussian prior and $q_\phi(\pi|\boldsymbol{x})$. Our task is to find the optimal I_{max} which leads to good synthesis as well as to good interpolation performance. By following common practice [9] we model $p(\boldsymbol{x}|\pi)$ as a parametric Laplace with constant standard deviation and assume that p_θ is deterministic in $\boldsymbol{x}$. By that, p_θ can be considered to be an image generator and the log likelihood can be rewritten as L_1 loss. In the next section, we introduce our used network architecture.

3.2. Network Architecture

The architecture of our network is based on the one from Alperovich et al.'s "Light field intrinsics using a deep encoder-decoder network" [1]. We used six blocks for the encoder, consisting of two three residual blocks of convolutions, whereas the first two keep dimensionality and the last reduces resolution by using a stride two convolution, and increases the feature size. It uses an input of nine aligned 48x48 pixel RGB patches. The decoder pathway exactly mirrors its encoder counterpart. The intermediate representation achieved after encoding the patches represents the bottleneck of our architecture and consists of 3x3x3x192 variables, reducing the input to about 10% of its size.

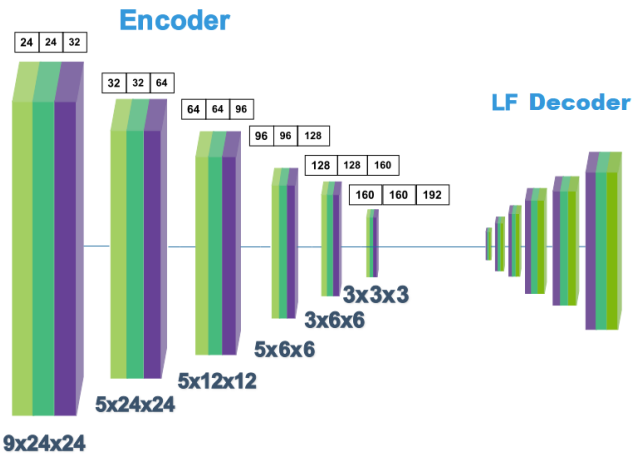


Figure 1: The encoder and decoder pathways are structured into six blocks of three residual blocks. The last layer in each residual block uses stride two to decrease resolution. The respective size of each layer can be seen on the bottom of the figure (viewpoint spatial coordinates). Feature depth (shown on top) is increased while going deeper in the architecture. The decoder is a mirrored copy of the encoder. This image was taken from "Light Field Intrinsics With a Deep Encoder-Decoder Network" by Alperovich et al. [1].

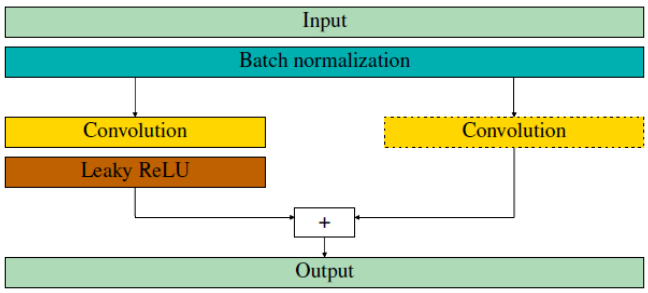


Figure 2: A single residual block. First batch normalization is applied, after which the input is passed onto a stride one or two convolution and a leaky ReLU. The second path is used to either pass the image through unchanged or also apply a stride two convolution (with kernel dimensionality 1×1) in order to match dimensionality, while using a learned kernel to increase feature size. For upsampling, a repeat operation is used instead of using stride two. Lastly both paths are added. This image was taken from "Light field intrinsics with a deep encoder-decoder network" by Alperovich et al. [1]

For the autoencoder, we simply used this architecture. To turn it into a variational autoencoder, we add a batch normalization and two convolutional layers after the encoder to predict μ and σ of the latent distribution, and then sample from it before passing the latent representation on to the de-

coder. For the autoencoder, we tried using both a L^1 and L^2 loss, for the variational autoencoder we used an L^1 loss and the KL-divergence in the way we introduced earlier.

When calculating the loss, we masked the first 8 pixels from the left and right edge for horizontal slices, and from the top and bottom for vertical slices. We also tried explicitly training the diagonal, here we masked 8 pixels from all edges when calculating the loss.

3.3. Viewpoint Interpolation

Given two representations of slices in latent space, L_1 and L_2, we tried interpolation between them as follows:

$$L = (\alpha L_1 + \beta L_2) / \gamma \tag{3}$$

In our case, $\gamma = \alpha + \beta$ seemed to be the appropriate normalization, as it produced the correct colors in the generated images. As we demonstrate later, this normalization only affected the color, and not the viewpoints, of the generated images.

To transform full resolution images, we split them into 48×48 pixel patches with a 16 pixel overlap and, dropping the outer border, assembled the generated images from the 16×16 pixel centre of the generated patches.

4. Results

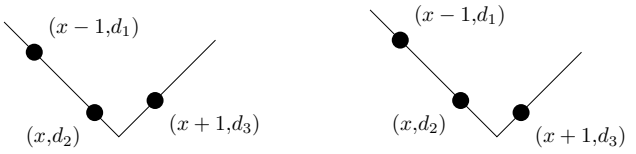
4.1. Dataset

For training and testing, we used the 4D Light Field Benchmark presented in [6]. We therefore had 20 training and 4 test light fields, each consisting of a 9×9 grid of 512px $\times$ 512px images. We trained our model with randomly cropped 48×48 pixel patches of the training light fields, using both the centre horizontal and vertical slices of the grid.

As data augmentation, we introduced random color jitter to the training patches. We also randomly rotated and mirrored them, resulting in eight distinct possibilities, while correcting the directions of the slices appropriately (such that the horizontal and vertical slices were always left to right and up to down, respectively).

4.2. Viewpoint Determination

To evaluate the generated images, we determined the position for every viewpoint we generated. For this, we assumed that the L_1-distance of the generated images to the original images is minimal for the original image that represents the same viewpoint; this is not necessarily true, as the original images are a lot sharper than the generated ones, but should nevertheless be a reasonable assumption. As we only have a discrete grid of original views, we further assumed that the L_1-distance increases linearly and symmetrically when the image is shifted along one axis. Using the distance of the three points along this axis that are closest to the minimum, we then determined its position (Figure 3a):



(a) Minimum within grid. (b) Minimum at the edge.

Figure 3: Position estimation. If $d_3 > d_1$, this needs to be flipped along the x-axis; (b) of course only works if the minimum is within the grid.

$$x_{\text{min}} = x + \frac{1}{2}\left(1 - \frac{d_1 - d_2}{d_3 - d_2}\right) \tag{4}$$

In the case where the minimum is on the edge of the grid (Figure 3b), this needs to be adjusted:

$$x_{\text{min}} = x + 1 - \frac{1}{2}\left(1 - \frac{d_2 - d_3}{d_1 - d_2}\right) \tag{5}$$

This was done independently for both axes. We also tried finding the viewpoint by finding the bilinear interpolation of four original images that resulted in the smallest L_1-distance. Both approaches worked well when testing them on the original images (dropping three quarters of the original images from the grid, so that the position is not immediately obvious).

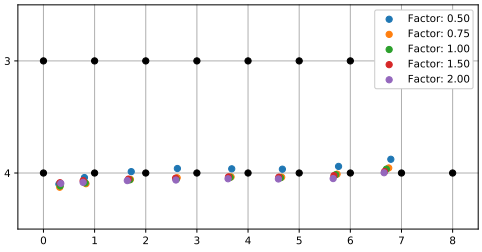
However, the bilinear approach failed when the images were less sharp than the original ones, i.e. when using either the generated images or smoothed original images; in this case, it always preferred a position at the centre of the images. In contrast, using the approach in Figure 3 worked well in both cases (Figure 11), therefore it was the one we ultimately used.

When testing the position inference on the original images, it worked less well on the edge of the grid, probably because assuming a linearly increasing L_1-difference over half of our light field is a bit of a stretch (for the missing images, our algorithm predicted a position outside of the grid). However, this is not a problem for the generated images, as the grid we use is considerably denser, and because almost no points are that close to the edge. Occasionally, there were points for which we could not determine a position (all of them close to the edge of the grid); they are simply omitted in the plots.

4.3. Autoencoder

The autoencoder we used for this section was trained on about 1.3 million cropped slices using an L_1-loss; we also tried using an L_2-loss, and the results were very similar. As can be seen in Figure 4, the images the autoencoder generated were quite good.

Figure 4: AE; the leftmost image of the horizontal slice for all test light fields; at the top is the original image, at the bottom the one our autoencoder generated.



(a) Inferred positions for the scaled horizontal slice.

(b) Scaled by 0.5, 1, and 2, respectively.

Figure 5: AE; results when scaling the latent representation by a factor before decoding; (a) shows the inferred positions, (b) examples for the resulting images.

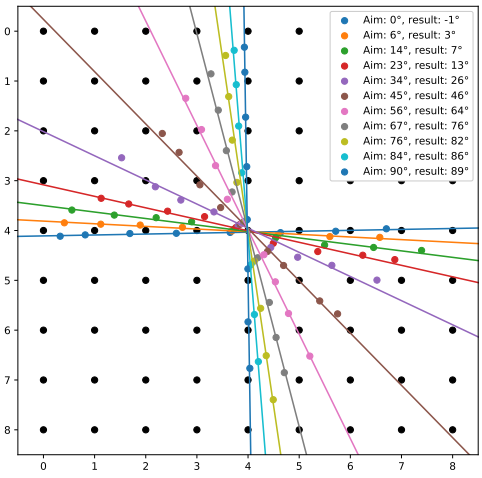
4.3.1 Scaling in Latent Space & Different Directions

Scaling the latent representation before decoding it did not change the viewpoints of the generated images (Figure 5a), however it did change the color intensity (Figure 5b). Of course, our position algorithm will break down if the colors of the image are changed sufficiently, but we also verified Figure 5a by looking at the generated images (as this is hard to see in static images, we provide appropriate GIFs in our GitHub repository). Therefore, the information about the viewpoints of the images is apparently not encoded in the magnitude of the latent representation.

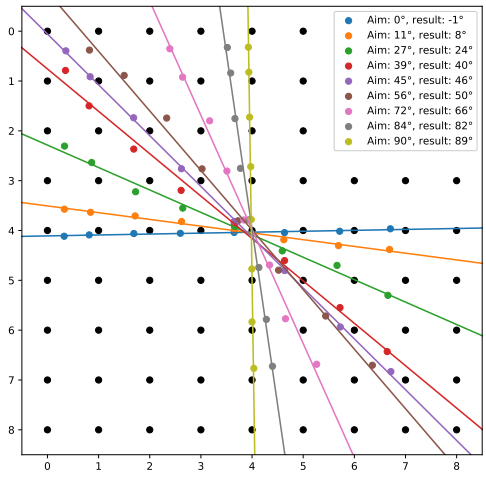
We were careful to only train the network on the left-right horizontal and up-down vertical slices of the light field. Nevertheless, it generalizes nicely to inverted directions (i.e. right-left horizontal etc.) and diagonals (Figure 13). We can also pass through shortened slices of the grid, generating the intermediate images by interpolation, so the network seems to genuinely infer the actual viewpoints for each image.

4.3.2 Interpolation in Latent Space

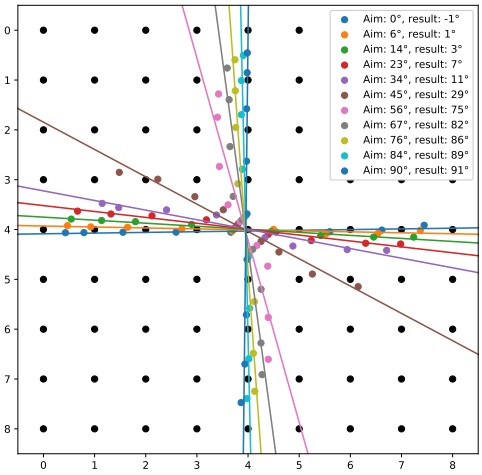
Interpolating in the latent space worked quite well (Figure 6). When interpolating between the horizontal and vertical slices, the resulting viewpoints essentially lie on a line between the corresponding original viewpoints (Figure 6a, Figure 6b). This is also the case when interpolating be-



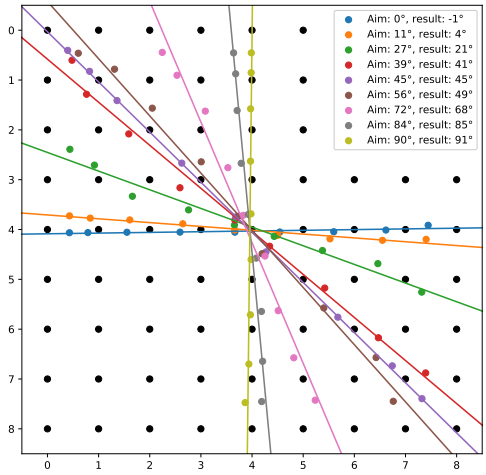
(a) First test light field, interpolation between horizontal and vertical slices.



(b) First test light field, interpolation between horizontal and diagonal, and diagonal and vertical slices.



(c) Third test light field, interpolation between horizontal and vertical slices.



(d) Third test light field, interpolation between horizontal and diagonal, and diagonal and vertical slices.

Figure 6: Interpolation in latent space using the autoencoder. (a) and (b) are on the first test light field, (c) and (d) on the third. For (a) and (c), the interpolation is done using only the horizontal and vertical slices, for (b) and (d) we interpolated between the horizontal and diagonal, and the diagonal and vertical slices.

tween the diagonal and horizontal or vertical slices (Figure 6c, Figure 6d). We also calculated the angle we would expect the interpolated slice to have, assuming that we are linearly interpolating between the x and y shifts of the original images (e.g. for $L = 0.8\,L_{\text{hor.}} + 0.2\,L_{\text{vert.}}$, the expected angle is $\arctan(0.2/0.8) = 14°$). For some light fields, such as the first test light field, the resulting angles were close to our expectation. For others, such as the third test light field, there was a larger discrepancy, but interpolating nevertheless worked.

We also tried explicitly training the sum in the latent space, i.e. training a batch of taking the sum of the horizon-

Figure 7: VAE; the leftmost image of the horizontal slice for all test light fields; at the top is the original image, at the bottom the one our autoencoder generated.

tal and vertical latent representations and decoding them, using the difference to the diagonal slice as loss, for every sixth batch. However, this did not change the results in any meaningful way; the generated slice still had the same viewpoints as in Figure 6.

Given that interpolation worked, and that we could pass our network different directions, we could generate a multitude of different viewpoints, using only the horizontal and vertical slices, or also the diagonals. To emphasize that the generated viewpoints are valid images, we again provide GIFs in our GitHub repository that essentially circle the whole light field, using mostly generated viewpoints.

Interestingly, interpolation was not limited to slices with the same centre viewpoint. For example, we could also interpolate between different horizontal slices (Figure 14), and even between a diagonal and the lowest horizontal slice (Figure 12).

4.4. Variational Autoencoder

Our variational autoencoder was trained on about 1.8 million cropped slices using an L_1-loss. We set the information budget I_max to 1000, which resulted in an average KL-loss of around 800, as we could see that a lower budget resulted in too blurry images. As can be seen in Figure 7, the images the model generated were acceptable, though somewhat more blurred than the ones the autoencoder generated.

Scaling in the latent space still only change the color of the images (Figure 16). Similarly, giving the network different directions works as well as for the autoencoder.

4.4.1 Interpolation in Latent Space

Interpolating in the latent space worked very similarly to the autoencoder, the interpolated viewpoints are still on a line between the original viewpoints (Figure 9). However,

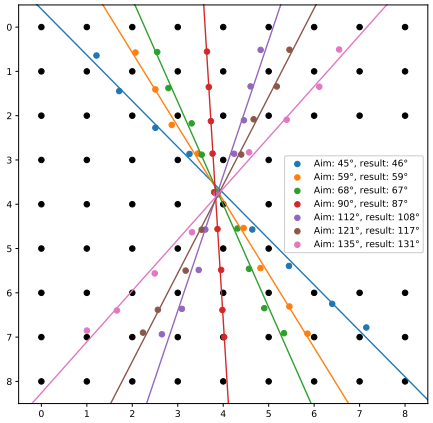
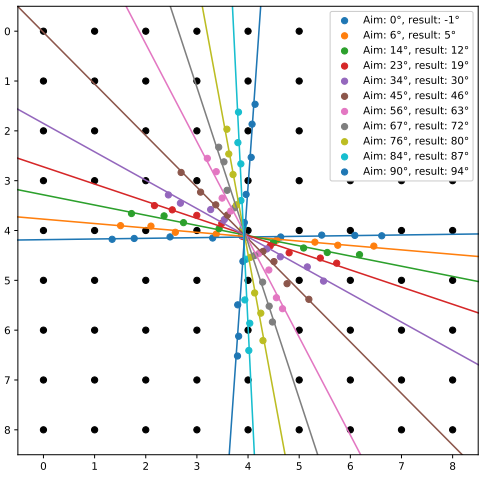
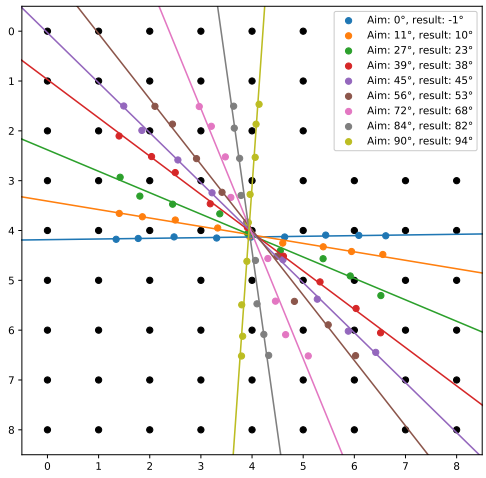


Figure 8: VAE; interpolating between diagonals.



(a) Third test light field, interpolation between horizontal and vertical slices.



(b) Third test light field, interpolation between horizontal and diagonal, and diagonal and vertical slices.

Figure 9: Interpolation in latent space using the variational autoencoder, for the third test light field.

it was much more stable and predictable than for the autoencoder. We now have a similar performance on all test images, and the generated angles were always quite close to our expectation. We again provide the same GIFs as for the autoencoder in our github repository.

Note that the interpolation was now stable enough that we could interpolate between diagonals (Figure 8), which did not work well for the autoencoder. We could even interpolate nicely between a diagonal and the lowest horizontal slice (Figure 10).

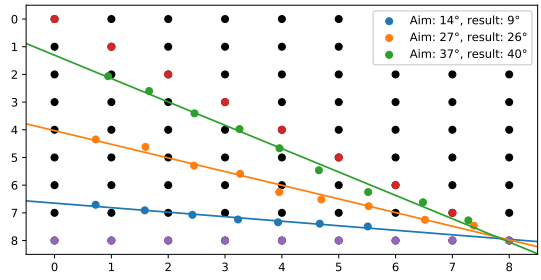


Figure 10: VAE; interpolating between a diagonal and the lowest horizontal slice of the first test light field.

5. Discussion

We showed that both for the autoencoder and the variational autoencoder, the viewpoints, or rather the shift between the different images in one slice, was independent of the normalization of the latent representation, and therefore encoded in some nontrivial way. The networks also generalized nicely to different directions of the slices, corresponding to a different direction of the camera shift, and diagonals, without explicitly training this.

To our surprise, we found that interpolation was possible even when using an normal autoencoder. We were able to interpolate continuously between most latent representations of light field slices, and obtained viewpoints that were essentially linear interpolations between the original viewpoints. However, the actual position of the interpolated viewpoints was not very well predictable, and performance varied strongly between different light fields. We were not able to train the network on decoding the full diagonal from the interpolation of the latent representations of the horizontal and vertical slices, which might potentially be due to the nontrivial encoding of the viewpoints in our particular network architecture.

We were able to show that by using variational autoencdoers the interpolation improved when compared to a normal autoencoder. This is coincides with theory since the regularization term for the variational autoencoder forces the network to arrange the viewpoints in a compacter format. We showed that the interpolations from the variational

autoencoder correspondeds closely to the resulting viewpoints. It worked consistently on all test images, indicating a better generalization to different light fields.

References

[1] Anna Alperovich et al. "Light Field Intrinsics With a Deep Encoder-Decoder Network". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[2] Christopher P. Burgess et al. *Understanding disentangling in β-VAE*. 2018. arXiv: 1804.03599 [stat.ML].

[3] Gaurav Chaurasia et al. "Depth Synthesis and Local Warps for Plausible Image-based Navigation". In: *ACM Transactions on Graphics* 32 (2013). to be presented at SIGGRAPH 2013. URL: http://www-sop.inria.fr/reves/Basilic/2013/CDSD13.

[4] Xi Chen et al. *Variational Lossy Autoencoder*. 2016. arXiv: 1611.02731 [cs.LG].

[5] Yang Chen, Martin Alain, and Aljosa Smolic. *Self-supervised Light Field View Synthesis Using Cycle Consistency*. 2020. arXiv: 2008.05084 [eess.IV].

[6] Katrin Honauer et al. "A dataset and evaluation methodology for depth estimation on 4D light fields". In: *Asian Conference on Computer Vision*. Springer. 2016.

[7] O. Johannsen et al. "A Taxonomy and Evaluation of Dense Light Field Depth Estimation Algorithms". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2017, pp. 1795–1812. DOI: 10.1109/CVPRW.2017.226.

[8] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. "Learning-Based View Synthesis for Light Field Cameras". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2016)* 35.6 (2016).

[9] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. arXiv: 1312.6114 [stat.ML].

[10] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*. 2014. arXiv: 1401.4082 [stat.ML].

[11] Heung-Yeung Shum, Shing Chan, and Sing Bing Kang. *Image-Based Rendering*. Jan. 2007. ISBN: 978-0-387-21113-8. DOI: 10.1007/978-0-387-32668-9.

[12] J. Yu. "A Light-Field Journey to Virtual Reality". In: *IEEE MultiMedia* 24.02 (Apr. 2017), pp. 104–112. ISSN: 1941-0166. DOI: 10.1109/MMUL.2017.24.

[13] Shengjia Zhao, Jiaming Song, and Stefano Ermon. *InfoVAE: Information Maximizing Variational Autoencoders*. 2017. arXiv: 1706.02262 [cs.LG].

[14] Tinghui Zhou et al. "Stereo Magnification: Learning View Synthesis using Multiplane Images". In: *SIGGRAPH*. 2018.

[15] H. Zhu, Q. Zhang, and Q. Wang. "4D Light Field Superpixel and Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6709–6717. DOI: 10.1109/CVPR.2017.710.

A. Further Plots

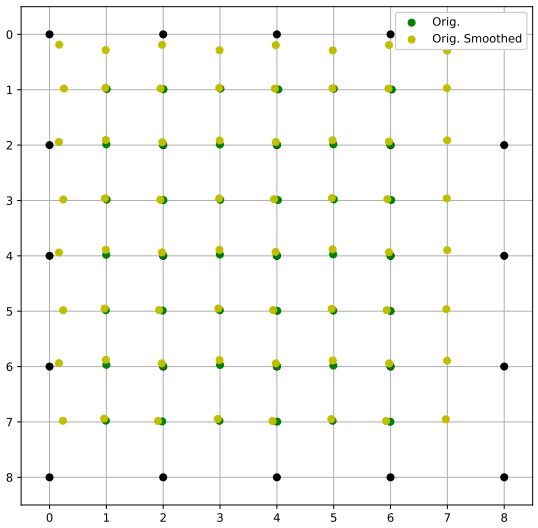


Figure 11: Testing our position algorithm on the original data (first test image); black points represent the original images we used as the ground truth (every second image along both axes), green points the inferred position of the original images, and yellow ones the inferred position of the original images after they were smoothed with a Gaussian filter.

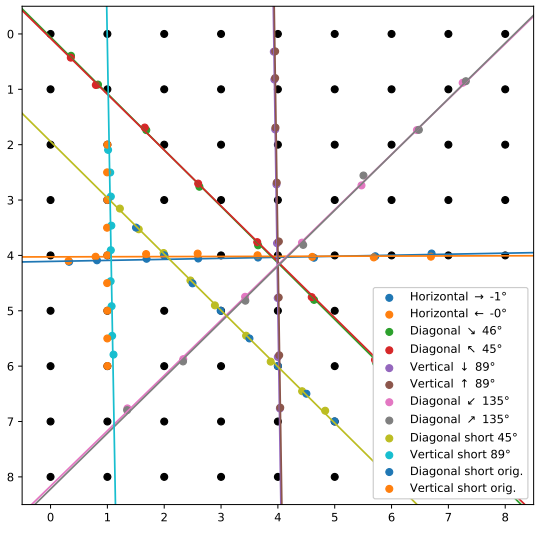


Figure 13: Passing various slices through the autoencoder.

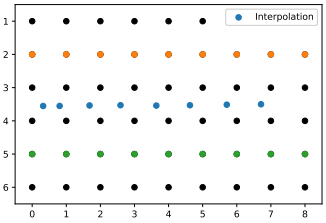


Figure 14: AE; interpolating between two horizontal slices for the first test light field.

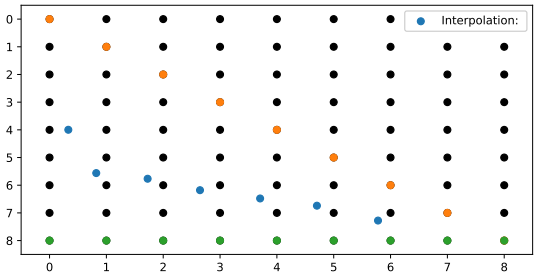


Figure 12: AE; interpolating between a diagonal and the lowest horizontal slice of the first test light field.

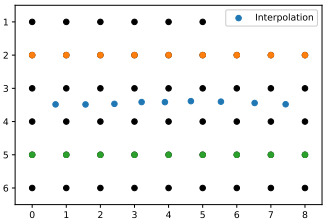
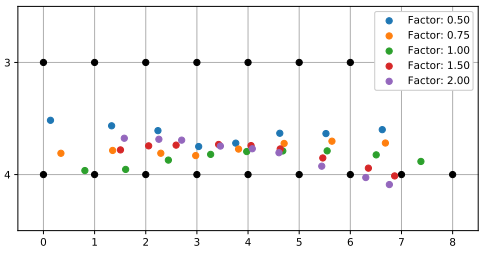
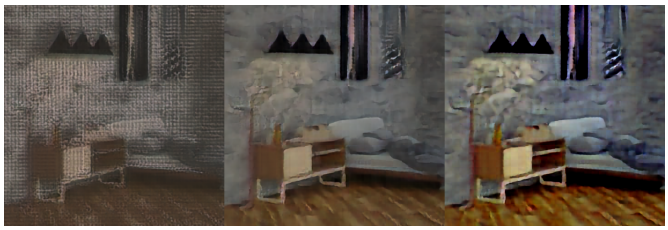


Figure 15: VAE; interpolating between two horizontal slices for the first test light field.



(a) Inferred positions for the scaled horizontal slice.



(b) Scaled by 0.5, 1, and 2, respectively.

Figure 16: VAE; results when scaling the latent representation by a factor before decoding; (a) shows the inferred positions, (b) examples for the resulting images.