Realtime Novel View Synthesis with Eigen-Texture Regression

Yuta Nakashima[1]
n-yuta@ids.osaka-u.ac.jp
Fumio Okura[2]
okura@am.sanken.osaka-u.ac.jp
Norihiko Kawai[3]
norihi-k@is.naist.jp
Hiroshi Kawasaki[4]
kawasaki@ait.kyushu-u.ac.jp
Ambrosio Blanco[5]
ambrob@microsoft.com
Katsushi Ikeuchi[5]
katsuike@microsoft.com

[1] Institute for Datability Science, Osaka University, Japan
[2] Institute of Scientific and Industrial Research, Osaka University, Japan
[3] Graduate School of Information Science, Nara Institute for Science and Technology, Japan
[4] Dep. Advanced Information Technology, Kyushu University, Japan
[5] Microsoft Research Asia, China

Abstract

Realtime novel view synthesis, which generates a novel view of a real object or scene in realtime, enjoys a wide range of applications including augmented reality, telepresence, and immersive telecommunication. Image-based rendering (IBR) with rough geometry can be done using only an off-the-shelf camera and thus can be used by many users. However, IBR from images in the wild (e.g., lighting condition changes or the scene contains objects with specular surfaces) has been a tough problem due to color discontinuity; IBR with rough geometry picks up appropriate images for a given viewpoint, but the image used for a rendering unit (a face or pixel) switches when the viewpoint moves, which may cause noticeable changes in color. We use the eigen-texture technique, which represents images for a certain face using a point in the eigenspace. We propose to regress a new point in this space, which moves smoothly, given a viewpoint so that we can generate an image whose color smoothly changes according to the point. Our regressor is based on a neural network with a single hidden layer and hyperbolic tangent nonlinearity. We demonstrate the advantages of our IBR approach using our own datasets as well as publicly available datasets for comparison.

1 Introduction

Realtime novel view synthesis (NVS) is a technique to generate a view of a certain real object or scene from an arbitrary viewpoint. It has a wide variety of applications [2, 24], such as augmented reality (AR), telepresence, and immersive telecommunication.

Image-based rendering (IBR) is an interesting approach for NVS with a photorealistic rendering quality. IBR basically represents an object (or a scene) with light paths that hit the

Figure 1: Example visual artifacts. Novel views are rendered from slightly different viewpoints. Significant color change in the middle novel view results in flickering in video.

object and then come into the camera without relying on the 3D geometry. View morphing [23, 27] is an early technique that mixes light paths captured in two different images to smoothly interpolate the viewpoint in-between them. Plenoptic functions [17] as well as light fields [16] and lumigraphs [10] model light rays at a certain 3D position and direction, which are extended to unburden the capturing process [4, 6]. For satisfactory rendering quality, these techniques require a vast amount of images, sometimes along with accurate 3D positions of the cameras. Other interesting approaches have been proposed so far in this field, *e.g.*, deep neural network-based NVS [9] and one for special scenes [14].

The view-dependent texture mapping (VDTM) technique by Debevec *et al.* [7, 8] is a promising approach that synthesizes photorealistic views from a feasible number of images, which can be viewed as a hybrid of model-based and image-based rendering. It uses rough geometry (a 3D mesh in most cases) as a proxy of the object's real shape and picks the appropriate color from the input images for the face of the mesh or a pixel in the resulting view. Due to the geometry, interpolation works well between different images. Varieties of VDTM's extensions have been proposed [5, 15, 18, 20, 22, 28].

A notable problem of VDTM is visual artifacts (*i.e.*, flickering) when the color of the same point is picked up from different images due to a moving viewpoint, as shown in Figure 1. This is because VDTM tries to find the image captured from the direction closest to the viewpoint. With automatic exposure or for objects with specular reflection, this problem causes severe degradation of the visual quality.

Some efforts have been made toward alleviating the problem of flickering. Original VDTM [8] mixes colors from not only one but several images captured from closest to view directions. This works well if the geometry is accurate and the lighting condition is stationary during image capturing. However, designing mixture weights that always gives a zero weight for an image just before it switches to another is tough due to discontinuity in the image selection. Buehler *et al.*'s technique [4] is an attempt in this direction, designing sophisticated image blending weights. Others [12, 15] rely on the expensive graphcut algorithm [3] and Poisson blending [21], which are not suitable for realtime rendering.

This paper proposes a novel technique for realtime NVS that offers smooth color changes even with input images in the wild (*i.e.*, uncontrolled and with specular reflection). Our idea is to use *eigen-texture* [19]. Eigen-texture is a low-dimensional representation of a set of small images. An image in the set is a point in the eigenspace, and we can interpolate in-between these images by just finding an appropriate point in the eigenspace. Using this representation, we can interpolate an image between different images without designing a weight function as in [4, 8] or expensive computation in the rendering process.

With this approach we replace image mixing with finding an appropriate point in the eigenspace. According to the IBR techniques [10, 16], a light ray at a certain 3D position is

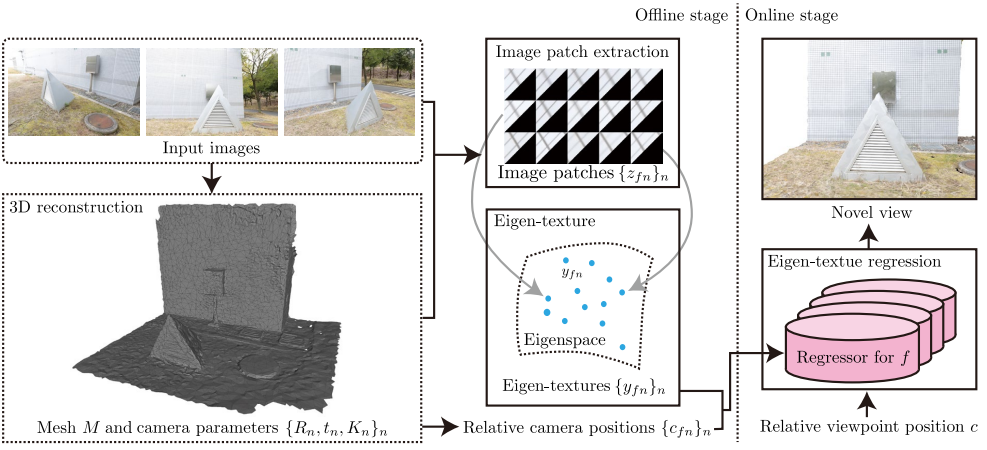

Figure 2: Overview of our system.

a function of its direction. Therefore, we synthesize the image for a certain face in the mesh by regressing a point in the eigenspace based on the direction to the camera. We use a neural network with a single hidden layer for this regression problem.

The main contribution of this paper is summarized as follows:

- We propose realtime NVS using eigen-texture [19] in order to avoid sudden changes in color. To synthesize an image from a novel viewpoint, we regress a point in the eigenspace. To the best of our knowledge, this is the first attempt to use eigen-texture for a freely moving viewpoint.
- For regression of a point in the eigenspace, we employ a neural network with a single hidden layer. This shallow network architecture works well for our regression problem. Thanks to the simplicity, it can be easily implemented in GPUs.
- We visually demonstrate the visual quality of our realtime NVS using eigen-texture over several datasets. To highlight the advantage, we compare it with some variants of VDTM techniques.

2 Overview

Figure 2 shows an overview of our NVS technique, consisting of the online and offline stages. In the offline stage, we estimate the intrinsic and extrinsic camera parameters as well as reconstruct a rough 3D geometry of a real object (it can be a scene, but we use an "object" for notation simplicity). We can manually build such a 3D geometry represented by a 3D mesh, or 3D reconstruction techniques work as well, such as the combination of VisualSFM [26] and CMPMVS [11]. Let $S = \{I_n | n = 1, \ldots, N\}$, $\{(K_n, R_n, t_n) | n = 1, \ldots, N\}$, and M be the set of input images, the camera parameters, and the 3D geometry, respectively, where $K_n \in \mathbb{R}^{3\times3}$, $R_n \in \mathbb{R}^{3\times3}$, and $t_n \in \mathbb{R}^3$ are intrinsic camera parameters, rotation matrix, and translation vector. Through the standard rendering pipeline, we generate depth map D_n of M for camera C_n that captured I_n using the camera parameters. Each face f in M is then projected to D_n. From images that passed visibility test for f (*i.e.*, whether f is visible in

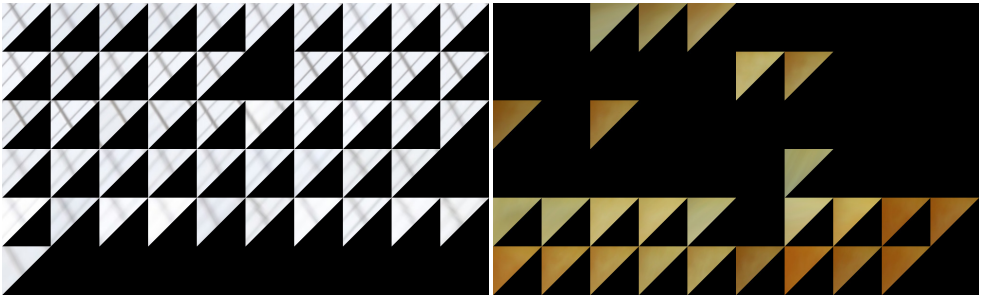

Figure 3: Some examples of image patches for certain faces. Image patch z_n is placed at the upper left triangle of the n-th block in the raster scan order. Black segments indicate that the face is not visible in I_n. The left image shows that the line patterns are not perfectly aligned and are blurry in some images. Right image shows that the color of the same face differs.

each of images or not), a set of image patches $Z_f = \{z_{fn}\}$ that correspond to f are extracted as shown in Figure 3. We find eigen-texture y_{fn} of z_{fn} in Z_f. The regressor for face f is also trained that maps a representation c of the viewpoint position to the eigen-texture y.

In the online stage, for given extrinsic camera parameter R and t with intrinsic camera parameter K of the viewpoint, the eigen-texture y_f for face f is regressed, and we reconstruct the corresponding image patch. The image patch is applied to f as texture to synthesize the novel view. The online process can be fully implemented on current GPUs.

3 Image Patch Extraction

Given 3D geometry M and camera parameters, we extract a set of image patches for each face in M from the images in S. Some faces are not visible in an image due to, *e.g.*, occlusion. We do visibility test, which is the same as [18], to get a set of images in which f is visible.

For visibility test, the standard rendering pipeline firstly generates the depth map D_n of M using R_n, t_n, and K_n. We deem face f is visible in I_n when all three vertices of f are visible. Let $v_{fi} \in \mathbb{R}^3$ denote the 3D position of vertex $i \in \{0,1,2\}$ of face f. We transform v_{fi} to camera C_n's coordinate system by

$$v^n_{fi} = R_n v_{fi} + t_n. \tag{1}$$

We then project v^n_{fi} onto D_n using K_n to identify the corresponding depth value d^n_{fi}. When this vertex is visible (*i.e.*, not occluded) in I_n, the depth value d^n_{fi} is sufficiently close to the third component of v^n_{fi}. Therefore, we judge that f is visible in I_n if $|(0\ 0\ 1)v^n_{fi} - d^n_{fi}| < \theta_{\text{vis}}$ is satisfied for all i. Threshold θ_{vis} is empirically determined based on the scale of M.

For face f, we extract the corresponding image patch z_{fn} from each image I_n in which f is visible to form set Z_f. In this work, we store the image patches as right isosceles triangle whose shorter edges are L pixels as in Figure 3. To extract the image patch, for each pixel in this triangle, we map it to the corresponding point on face f and then project the point to I_n using the camera parameters. Let α_1 and α_2 be barycentric coordinates of pixel k in the image patch, where pixel k's position $p'_k \in \mathbb{R}^2$ is given using vectors u'_1 and u'_2 representing the two same-length edges of the image patch z_{fn} by

$$p'_k = \alpha_1 u'_1 + \alpha_2 u'_2 \quad (\alpha_1 + \alpha_2 \leq 1). \tag{2}$$

The corresponding point p_k on f is given by $p_k = \alpha_1 u_1 + \alpha_2 u_2$, where u_1 and u_2 are the corresponding edges of f. Point p_k is then projected to image I_n using camera C_n's parameters as in Equation 1. We use bilinear interpolation to get the pixel colors of non-integer positions.

4 Eigen-texture

The eigen-texture method [19] is a technique to embed image patches into a low-dimensional space, which is identified by eigen-decomposition. Let $\tilde{z}_n$ be a vectorization of image patch z_n and $\bar{z}$ be the average over Z (we omit subscript f for notation simplicity). All image patches are centerized by $\tilde{z}_n - \bar{z}$ and then are concatenated into matrix $\tilde{Z}$, where each column is $\tilde{z}_n - \bar{z}$. We can factorize $\tilde{Z}\tilde{Z}^\top$ with eigen-decomposition:

$$\tilde{Z}\tilde{Z}^\top e_j = \lambda_j e_j, \tag{3}$$

where e_j and λ_j are the j-th eigenvector and eigenvalue, respectively. The low-dimensional eigenspace is spanned by top J eigenvectors.

An arbitrary image patch z can be embedded into this eigenspace by

$$y = E^\top (z - \bar{z}), \tag{4}$$

where E is a matrix whose columns are a set of largest eigenvectors. We can also reconstruct image patch z from y with

$$z = Ey + \bar{z}. \tag{5}$$

5 Eigen-Texture Regression

Equation (5) implies that we can synthesize an image patch from an arbitrary point in the eigenspace. We assume that the light sources in the scene are fixed during image capturing. Under this assumption, the image patch is a function of the viewpoint position relative to the corresponding face. This means that we can synthesize an appropriate image patch for any viewpoint if we can regress a point in the eigenspace for that viewpoint. We do this with a shallow neural network-based regressor that maps a certain representation of the viewpoint to a point in the eigenspace.

As the viewpoint representation, we use the position of the viewpoint in a coordinate system defined on each face. Using u_1 and u_2 that represent two edges of face f, we can compute a normal vector $a_0 \in \mathbb{R}^3$ of f by

$$a_0 = \frac{u_1 \times u_2}{\|u_1\| \, \|u_2\|}, \tag{6}$$

where the operator "$\times$" is the cross product. We define f's coordinate system by taking a_0, $a_1 = u_1/\|u_1\|$, and $a_2 = a_0 \times n_1$ as the axes as well as $o = \sum_i v_i/3$ as the origin. In this coordinate system, we can represent the viewpoint position $c \in \mathbb{R}^3$ by

$$c = A^\top (-R^\top t - o), \tag{7}$$

where $A = (a_0 \;\; a_1 \;\; a_2)$, R and t are the rotation matrix and translation vector of the viewpoint.

The architecture of our neural network-based regressor is a single hidden-layer network with the hyperbolic tangent nonlinearity given by

$$\tilde{y}(c) = W_2 \tanh(W_1 c + b_1) + b_2, \tag{8}$$

where W_1 is in $\mathbb{R}^{U\times 3}$, W_2 in $\mathbb{R}^{J\times U}$, and U the number of the hidden units. The loss

$$\ell(\{y_n\},\{c_n\}) = \sum_n \|y_n - \tilde{y}(c_n)\|^2 \tag{9}$$

for each face is separately minimized during the training, where c_n is the camera C_n's position relative to f given by Equation (7) and y_n is the point to which image patch z_n is embedded. The regressor is trained with the gradient descent algorithm. We employ weight decay for regularization. We normalized c_n so that each of its elements has unit variance.

6 Online Stage

The online stage renders the object given a viewpoint (*i.e.*, rotation matrix R and translation vector t, as well as intrinsic camera parameter K) in realtime. For face f, we compute the relative viewpoint position c with Equation (7). Then the point y in the low-dimensional space is regressed via Equation (8). To obtain y, we do the inverse of the normalization applied to y_{nf}. We synthesize image patch z corresponding to y with Equation (5). Finally image patch z is applied to face f as a texture. Thanks to the light-weight architecture of our regressor, the online stage can be implemented in a GPU to achieve realtime rendering.

7 Datasets and Implementation Details

We tested our realtime NVS using two datasets from [12] (*i.e.*, triangle pyramid and penguin datasets). From the images in these datasets, we estimated the camera parameters and reconstructed 3D meshes using VisualSFM [26] and CMPMVS [11]. The numbers of faces in these meshes were reduced to 10K (triangle pyramid) and 5K (penguin) using quadratic edge collapse decimation. We also used a publicly available Buddha dataset in [1, 25]. We reduced the number of faces from over 2M to 5K for reducing the time for regressor training. Refer to the supplementary material for more details on these datasets.

Figure 4 shows histograms of the contribution ratios for $J = 1$, 5, and 10. For $J = 1$, the contribution ratio has a long-tail distribution. The distribution gradually gets concentrated as J increases. This demonstrates that $J = 10$ covers most variations in image patches. To secure the visual quality, we used $J = 10$ in the following section; however, smaller J might be sufficient. The number U of our regressor's hidden units is 10. The weight of weight decay was set to 0.1. We employed an variant of the gradient descent algorithm, Adam [13], but gradients were computed for all image patches at once (*i.e.*, the batch size equals to the number of visible image patches) since the number of image patches is not very large. The network parameters were updated for 30,000 times (*i.e.*, the number of epochs is 30,000). Some faces do not have enough image patches and thus $J = 10$ eigenvectors could not be obtained. We instead filled the rest of the entries of E with zeros.

As a baseline, we employed simple VDTM, which picks up one most suitable image for each pixel in the novel view and copies the corresponding color. This technique basically gives a novel view with spatial and temporal color discontinuities. Our second baseline is

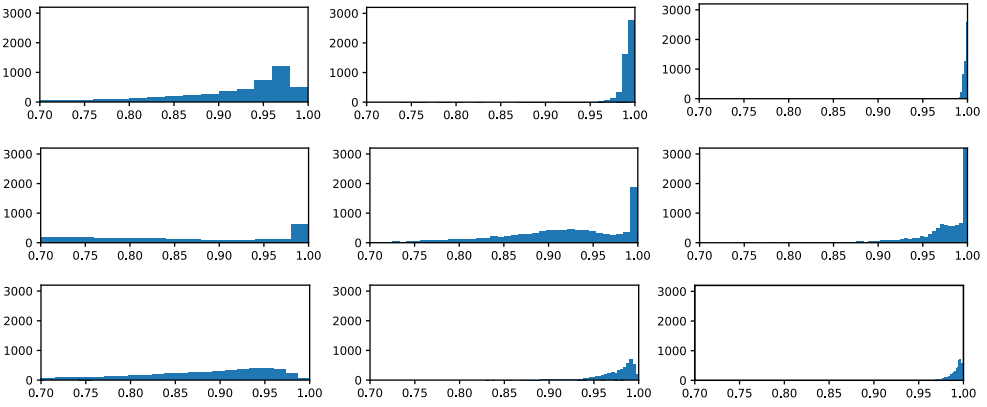

Figure 4: Histograms of contribution ratios for penguin (top), triangle pyramid (middle), and Buddha (bottom) datasets for $J = 1$ (left), 5 (middle), and 10 (right). When $J = 1$ the contribution ratios are widely distributed. As J increases, the distribution gets more compact.

VDTM with texture blending. This is the most similar to the original VDTM technique [8]. We find three most suitable images from images and use their mixture for the pixel. The mixture weight is based on the cosine similarity s between the vectors (i) from the corresponding point in the mesh to camera C_n, and (ii) from the same point to the viewpoint. As the third baseline, we add temporal blending to the second baseline (VDTM with temporal blending). Let g_t be a certain pixel value in the current novel view before temporal blending and g'_{t-1} the corresponding pixel value in previous one after temporal blending, where the corresponding pixel is found based on the depth map. We compute the current novel view's pixel value g'_t by mixing them by *i.e.*, $g'_t = (g_t + g'_{t-1})/2$.

8 Visual Results and Comparisons

Figures 5, 6, and 7 show an example novel view synthesized from moving viewpoints (best viewed in supplementary video). In these figures, we also show novel views synthesized with mean image patches $\{\bar{z}_f\}_f$ (mean views) and the difference between our technique's novel views and the mean views (difference views).

The triangle pyramid dataset has specular reflection on the silver box on the wall. In the baselines, the color on that surface changed when the image switched. This may give us non-realistic perception. Our technique again gradually changes the color. In the difference view, we can see that the surfaces of the triangle pyramid change a lot. This is because the object has a weak specular reflection component, which is not reproduced in the mean view. This component also causes significant color changes in the baselines, while ours handled it.

The penguin dataset has fine texture of the stuffed penguin's fluffy surface. This texture is preserved in our technique, even though it comes from low-dimensional space (compare it with the mean view). Simple VDTM has clear edges in the penguin's back due to image switching. Such discontinuity is not visible in VDTM with texture blending and temporal blending since texture and temporal blending works well. However, this is still noticeable in video (refer to our supplementary video). We can also observe that the anisotropic reflection on the fluffy surface, which gradually changes the color, is preserved, while it is lost in the

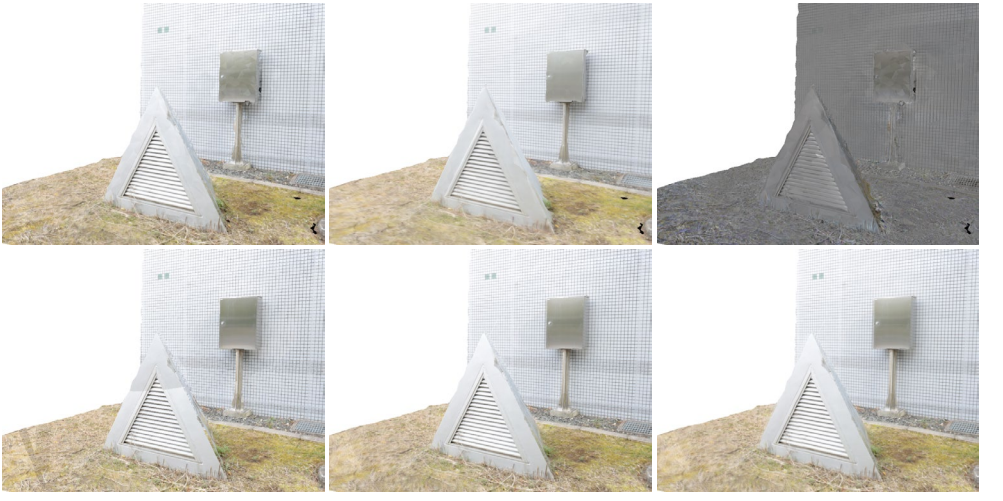

Figure 5: Novel views from the triangle pyramid dataset. From top-left to bottom-right: Our eigen-texture-based technique, and its mean and difference views, simple VDTM, VDTM with texture blending, and VDTM with temporal blending.

Figure 6: Novel views from the penguin dataset. From left to right: Our eigen-texture-based technique, and its mean and difference views, simple VDTM, VDTM with texture blending, and VDTM with temporal blending.

Figure 7: Novel views from the Buddha dataset.

baselines.

The Buddha dataset is easy because it has more images and there only is diffuse reflection. Yet VDTM and ours offers clear difference. We guess the light source moved with the camera while capturing because the luminance around the upper arm and the body changes

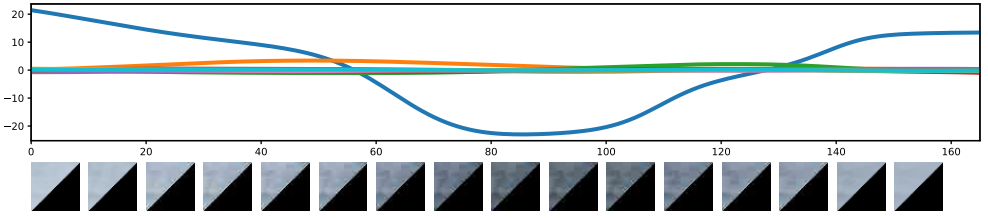

Figure 8: Evolutions of $\tilde{y}$ when viewpoint moves (top) and image patches (bottom).

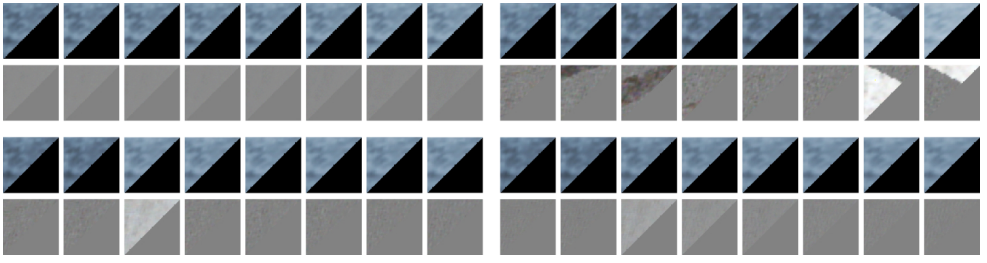

Figure 9: Image patches when viewpoint moves and the differences from the previous one. From top-left to bottom-right: ours, simple VDTM, VDTM with texture blending, and VDTM with temporal blending. We enhanced the contrast for visibility.

as the camera moves around. As in the difference view in Figure 7, ours reproduced this change. This is because the light source moved along with the camera, which is a special case that our technique works with moving light sources. In contrast, the VDTM-based techniques make temporally discontinuous changes.

Figure 8 shows the evolution of each component of regressed point $\tilde{y}$. This figure indicates that the most variations in this face (one in the penguin dataset) is described by the first eigenvector (the cyan line), which is major according to Figure 4. Since the curves are smooth, resulting image patches also smoothly change. Figure 9 shows the reconstructed image patches and their difference from the previous image patch around the 110-th time unit in Figure 8. Ours constantly and gradually changes the color, while the VDTM-based techniques give noticeable spikes.

Performance. The online stage ran at 60 fps (capped by the display's refresh rate), 56 fps, and 21 fps for the Buddha, penguin, and triangle pyramid datasets on average using CUDA implementation on a NVIDIA Titan X Pascal GPU (12 GB VRAM, 3584 cores).

Limitations. Triangular shapes are noticeable in our novel views as eigen-textures/regressors could not give image patches sufficiently close to input images. One of the reasons is that the number of image patches is too small due to occlusion to cover all variations. This is an inherent problem of NVS or IBR, and only the solution may be to add more images. Another possible reason is independent training of regressors. Our regressor training is independent for each face, and any smoothness terms for adjacent faces are not used at all. We can handle this problem by incorporating smoothness terms in the loss function; however, training will be much more expensive. Another remedy can be to adjust colors in the online stage. We can employ Poisson blending [21] to alleviate such artifacts at the cost of the frame rate.

Extrapolation is another weak point of our technique, which is shared by most IBR techniques. In our case, the regressor's output is usually unstable when the viewpoint is far from

any cameras. A countermeasure is to train the regressor to give zero (and thus gives the mean image patch) when it is far from cameras. Resulting novel views are not faithful to the real object, but the visual quality will be better.

9 Conclusion

This paper proposed eigen-texture-based NVS. Being different from the original eigen-texture method [19], ours can interpolate image patches for arbitrary viewpoint using neural network-based regressors. Our experimental results demonstrated that it could reproduce specular reflection on metallic surfaces and anisotropic reflection on fluffy surfaces, as well as view-dependent lighting. Temporal smoothness of resulting novel views is the most important aspect of our technique, which basically is infeasible for VDTM in an uncontrolled image capturing process or with anisotropic reflection surfaces. Our future work includes employing smoothness terms among adjacent faces in the loss function or Poisson blending in the online stage. Since our technique represents all image patches for a face by a fixed-size set of eigenvectors, we can increase the number of input images without any cost in the online stage. We will test this aspect as well.

References

[1] *Image-Based Modeling and Rendering Benchmark*, 2017 (accessed May 1, 2017). https://ibmr-benchmark.gcc.informatik.tu-darmstadt.de/.

[2] R. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[3] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[4] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured lumigraph rendering. In *Proc. ACM SIGGRAPH*, pages 425–432, 2001.

[5] G. Chaurasia, S. Duchêne, O. S. Hornung, and G. Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Trans. Graphics*, 32(3):30:1–30:12, 2013.

[6] A. Davis, M. Levoy, and F. Durand. Unstructured light fields. *Computer Graphics Forum*, 31(2):305–314, 2012.

[7] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proc. ACM SIGGRAPH*, pages 11–20, 1996.

[8] P. E. Debevec, Y. Yu, and G. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Proc. Eurographics Workshop on Rendering (EGWR)*, pages 105–116, 1998.

[9] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. DeepStereo: Learning to predict new views from the world's imagery. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 5515–5524, 2016.

[10] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Proc. ACM SIGGRAPH*, pages 43–54, 1996.

[11] M. Jancosek and T. Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 3121–3128, 2011.

[12] K. Katagiri, Y. Nakashima, T. Sato, and N. Yokoya. Novel view synthesis based on view-dependent texture mapping with geometry-aware color continuity. *Trans. Virtual Reality Society of Japan*, 21(1):153–162, 2016.

[13] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. Int. Conf. Learning Representations (ICLR)*, 15 pages, 2015.

[14] J. Kopf, F. Langguth, D. Scharstein, R. Szeliski, and M. Goesele. Image-based rendering in the gradient domain. *ACM Trans. Graphics*, 32(6):199:1–199:9, 2013.

[15] J. Kopf, M. F. Cohen, and R. Szeliski. First-person hyper-lapse videos. *ACM Trans. Graphics*, 33(4):78:1–78:10, 2014.

[16] M. Levoy and P. Hanrahan. Light field rendering. In *Proc. ACM SIGGRAPH*, pages 31–42, 1996.

[17] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Proc. ACM SIGGRAPH*, pages 39–46, 1995.

[18] Y. Nakashima, Y. Uno, N. Kawai, T. Sato, and N. Yokoya. AR image generation using view-dependent geometry modification and texture mapping. *Virtual Reality*, 19(2): 83–94, 2015.

[19] K. Nishino, Y. Sato, and K. Ikeuchi. Eigen-texture method: Appearance compression and synthesis based on a 3D model. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(11):1257–1265, 2002.

[20] S. Nobuhara, W. Ning, and T. Matsuyama. A real-time view-dependent shape optimization for high quality free-viewpoint rendering of 3D video. In *Proc. Int. Conf. 3D Vision (3DV)*, pages 665–672, 2014.

[21] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Trans. Graphics (Proc. ACM SIGGRAPH)*, 22(3):313–318, 2003.

[22] D. Porquet, J.-M. Dischler, and D. Ghazanfarpour. Realtime high-quality view-dependent texture mapping using per-pixel visibility. In *Proc. Int. Conf. Computer Graphics and Interactive Techniques (GRAPHITE)*, pages 213–220, 2005.

[23] S. M. Seitz and C. R. Dyer. View morphing. In *Proc. ACM SIGGRAPH*, pages 21–30, 1996.

[24] M. Tanimoto, M. P. Tehrani, T. Fujii, and T. Yendo. Free-viewpoint TV. *IEEE Signal Processing Magazine*, 28(1):67–76, 2011.

[25] M. Waechter, M. Beljan, S. Fuhrmann, N. Moehrle, J. Kopf, and M. Goesele. Virtual rephotography: Novel view prediction error for 3D reconstruction. *ACM Trans. Graphics*, 36(1):8:1–8:11, 2017.

[26] C. Wu. Towards linear-time incremental structure from motion. In *Proc. Int. Conf. 3D Vision (3DV)*, pages 127–134, 2013.

[27] J. Xiao and M. Shah. From images to video: View morphing of three images. In *Proc. Vision, Modeling and Visualization (VMV)*, pages 495–502, 2003.

[28] Q.-Y. Zhou and V. Koltun. Color map optimization for 3D reconstruction with consumer depth cameras. *ACM Trans. Graphics*, 33(4):155:1–155:10, 2014.