Pacific Graphics 2015
N. J. Mitra, J. Stam, and K. Xu
(Guest Editors)

Volume 34 (2015), Number 7

4D Model Flow: Precomputed Appearance Alignment for Real-time 4D Video Interpolation

Dan Casas [1,2] Christian Richardt [2,3] John Collomosse [1] Christian Theobalt [2] Adrian Hilton [1]

[1] University of Surrey [2] MPI Informatik [3] Intel Visual Computing Institute

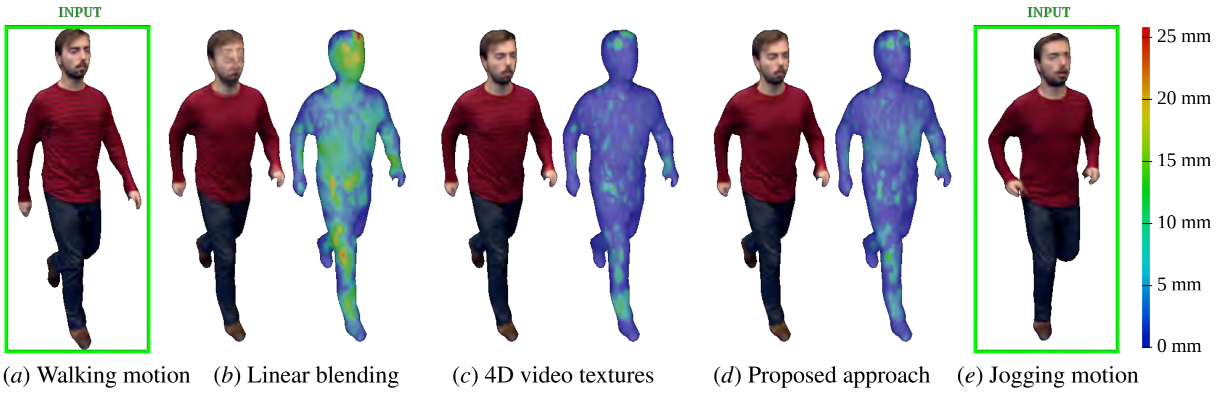

Figure 1: Appearance interpolation between two 4D videos of *walking* and *jogging* motions (*a* and *e*, highlighted), at the half-way position, effectively synthesizing a very fast walk motion (notice the change in pose in *b*–*d* compared to *a* and *e*). (*b*) Linear appearance interpolation results in severe ghosting artifacts (e.g. on the face). (*c*) Online appearance alignment using 4D video textures [CVCH14]. (*d*) Our appearance alignment using precomputed 4D model flow has comparable visual quality to (*c*), but runs an order of magnitude faster. The heat maps visualize the quality of appearance alignment using the optical flow between the two warped input appearances (*a* and *e*) before they are blended (see Section 6.1 for details).

Abstract

We introduce the concept of 4D model flow for the precomputed alignment of dynamic surface appearance across 4D video sequences of different motions reconstructed from multi-view video. Precomputed 4D model flow allows the efficient parametrization of surface appearance from the captured videos, which enables efficient real-time rendering of interpolated 4D video sequences whilst accurately reproducing visual dynamics, even when using a coarse underlying geometry. We estimate the 4D model flow using an image-based approach that is guided by available geometry proxies. We propose a novel representation in surface texture space for efficient storage and online parametric interpolation of dynamic appearance. Our 4D model flow overcomes previous requirements for computationally expensive online optical flow computation for data-driven alignment of dynamic surface appearance by precomputing the appearance alignment. This leads to an efficient rendering technique that enables the online interpolation between 4D videos in real time, from arbitrary viewpoints and with visual quality comparable to the state of the art.

1. Introduction

Modern performance capture methods enable the reconstruction of moving, textured surfaces with an underlying temporally coherent geometry that deforms over time, so-called *4D videos* [BHKH13, BHB*11, dAST*08]. These models can be textured classically [TFB14], projectively [BBM*01] or by free-viewpoint rendering that captures view-dependent details [DYB98]. Each recorded 4D video can easily be played back, but not modified, since geometry, motion and appearance need to be modified coherently.

Recently, the first methods were proposed that enable the synthesis of new dynamic shape sequences by interpolating between discrete captured motions in a space of shape sequences. At first, this was demonstrated for geometry only [HHS09, CTGH13]. However, synthesizing an interpolated textural appearance is a different and challenging problem in its own right. Initial attempts combined closest-pose retrieval from a database of poses with image warping to align the textures [XLS*11], while more recent work uses a parameterized motion space with online optical flow alignment at render time to combine the dynamic appearance from multiple sequences [CVCH14]. This enables plausible texture synthesis for interpolated poses (such as in Figure 1c), but online optical flow computation at render time is computationally expensive, which prevents its use in real-time applications. Precomputation of dense surface correspondences is desired; however, existing approaches for surface tracking [BHKH13, dAST*08, BHPS10] deform the underying geometry proxy to store the correspondences, limiting the acuracy by the mesh resolution.

We contribute in this paper by introducing the concept of the *4D model flow*, the dense appearance correspondence field between any two 4D models (the frames of 4D videos), for example between different poses such as walking and running motions of a human character.

In order to synthesize the appearance of intermediate poses across sequences, one needs to compute the 4D model flow, which brings the surface appearance of different poses into correspondence. This correspondence is defined in terms of visual appearance similarity on the surface, and not the underlying geometry itself, as is the case in scene flow [VBR*05]. While scene flow aims to compute geometric correspondence between poses, and often uses appearance for this goal, our 4D model flow explicitly computes the “appearance motion” between 4D video models. This is because surface appearance changes over time, such as clothes and hair move dynamically across surfaces during motions. Using the 4D model flow, one can create the dynamic surface appearance of an arbitrary interpolated 4D model by simply indexing into the flow at interpolated locations, and retrieving appearance information from the endpoints (4D models).

Note that 4D model flow provides accurate per-pixel appearance correspondences regardless of the underlying mesh resolution, which enables fine-detail appearance alignment across 4D models even using coarse geometries. Such detailed alignment cannot be achieved by mesh tessellation of the 4D model, for example using 3D displacement maps, because triangle subdivision, based on barycentric interpolation, inherently loses the correct correspondence within the triangle. Furthermore, current approaches for full-body 4D video reconstruction [BHKH13, dAST*08] do not achieve accurate, high-resolution geometry registration, but only provide a relatively coarse but temporally coherent representation of the scene. Despite this limitation, our 4D model flow exploits the known coarse geometry alignment for detailed appearance alignment.

Here, we show how to estimate the 4D model flow from a set of meshes and multi-view images. For this, we use image-based correspondence finding, guided by 3D geometry proxies. Since the 4D model flow is defined on the mesh surface, we show how it can be stored compactly in a UV map, which is a view-independent and efficient way for indexing into it. This enables real-time 4D video interpolation, as shown in Figure 2 and our supplementary video, with comparable visual fidelity to previous approaches but at 10× higher frame rates.

2. Related Work

Free-viewpoint playback of recorded performances has received considerable attention. Several complementary research directions have been explored, including multi-view depth estimation and view interpolation [ZKU*04, EDDM*08] and reconstruction of 3D model sequences from multi-view videos [SH07, VBMP08]. More recently, mesh tracking using geometric [ZBH12] and appearance cues [BHKH13, BHB*11] has enabled these per-frame mesh models to be conformed to a single deforming mesh over time (so-called 4D video). The resulting texture parametrization over such meshes was subsequently leveraged to extract a single static texture map, capturing a Lambertian model of appearance [TFB14], and harnessed for blending mesh sequences [CTGH13].

The small size and low complexity of static texture maps are ideal for mesh animation, but they lack the realism of free-viewpoint video that is achieved by sampling multi-view videos in a view-dependent manner at render time. Video textures use intra-frame correspondence in a monocular video, and a temporal frame re-ordering method to create new monocular video where an object follows a path designed by a user or a captured skeleton motion [FNZ*09]. A full spatio-temporal mapping between all frames of two monocular video sequences was computed in [LLN*14] to enable plausible monocular video morphing under non-rigid motion. Our approach enables smooth variation of viewpoint and interpolation between 4D videos at the same time. Dense correspondence techniques can be used to align view-dependent textures, as they achieve sufficient accuracy for this sort of image alignment task [LYT11, SRB14]. Eisemann et al.’s *floating textures* [EDDM*08] explored the use of optical flow to remove the ghosting artifacts caused by blending misaligned view-dependent projective textures on meshes. Zhou et al. [ZK14] follow a similar approach and jointly optimize camera positions as well as image-based warps to make projective textures align well on an coarse and imperfect static geometry obtained with an RGB-D camera and KinectFusion.

These approaches are inherently limited to be used for the acquired geometric proxy and can not be applied for

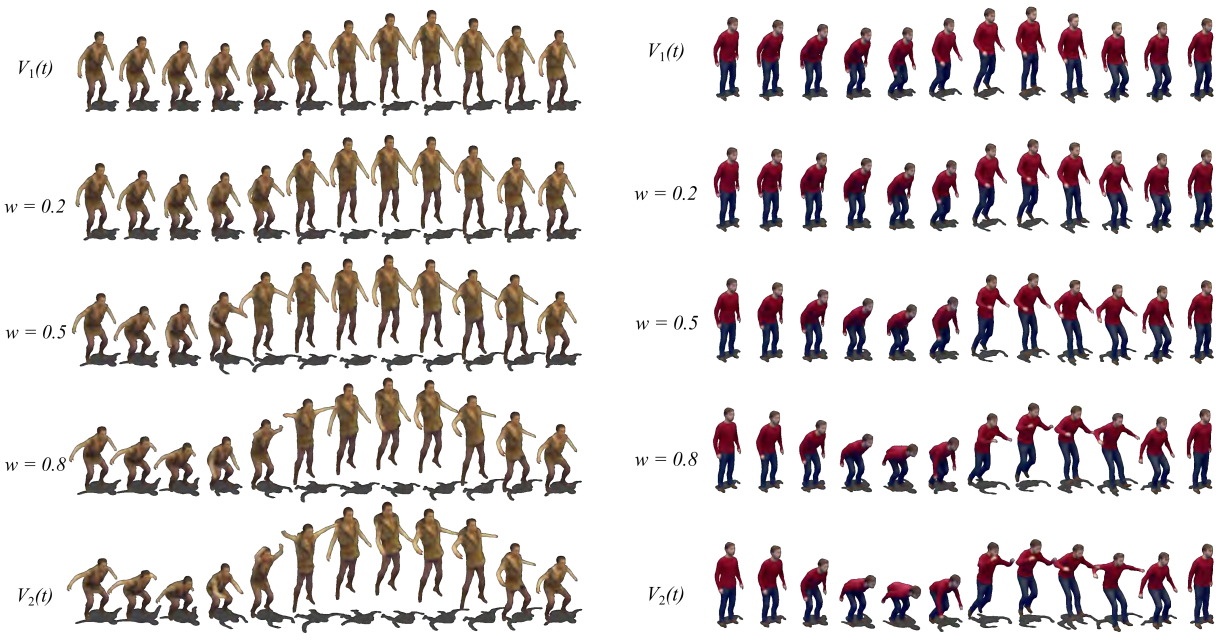

Figure 2: 4D video interpolation results between small jumps (top) and large jumps (bottom), for two different datasets (left/right). Rows 2–4 use different interpolation factors $w \in \{0.2, 0.5, 0.8\}$, where each model is interpolated accordingly between the corresponding input models in the first and last row. Note how the body pose changes between the rows. For better visualization, we normalized the length of input sequences (i.e number of frames) and show only a selection of frames. All frames are rendered individually from the same viewpoint, not considering any global translation resulting from the jumping motion itself.

appearance alignment between different 4D models. Casas et al. introduced *4D video textures* [CVCH14] to address this issue by blending the 4D video appearance of different 4D videos at run time, which complements earlier work on blending 4D mesh geometry only [CTGH13]. In their work, optical flow alignment is used to interpolate between view-dependent textures across sequences. Texture alignment is determined at render time using free-viewpoint projection of the source multi-view videos. Recently, Volino et al. [VCCH14] presented a novel efficient representation to alleviate the bottleneck of streaming and storing multi-view videos on graphics hardware, which combined with previous work [CVCH14] opens up the opportunity for interactive character animation from 4D video performance capture. However, interpolation across 4D video sequences remains limited by computationally expensive dense optical flow estimation at run time, hindering the practical uses of 4D video in heavily resource-contended applications such as computer games. This paper addresses this shortcoming by parametrizing and encoding appearance alignment in multi-view videos using precomputed 4D model flows.

Our proposed 4D model flow might at first seem similar to 3D scene flow, “the 3D motion field describing the motion of every visible 3D point between two time steps” [QBDC14]. Like our 4D model flow, scene flow is usually computed from multiple optical flows, for example using voxel coloring [VBR*05], within a variational framework [PKF07, VBZ*10, WBV*11], or with piecewise rigid motion priors [VSR13]. However, while scene flow expresses the motion within a scene over time, i.e. aims to recover purely geometric correspondence, model flow expresses the “appearance motion” between different scenes over time. Specifically, our 4D model flow is the 3D motion field describing the appearance correspondence of every visible 3D point between two 4D video models (hence “4D” in the name).

3. Overview

This paper introduces the concept of 4D model flow, the dense correspondence across 4D video models, for aligning varying appearances (Section 4). Precomputed 4D model flows can be stored efficiently and enable real-time interpolation of 4D video models from arbitrary viewpoints. For the first time, this allows interactive rendering of parametric, view-dependent and video-realistic animations from a sparse set of captured 4D video models, without the need for texture alignment computations at run time.

Our computational approach comprises two main steps (see Figure 3). First, we compute the 4D model flow for each frame of the output interpolated animation. We map the ap-

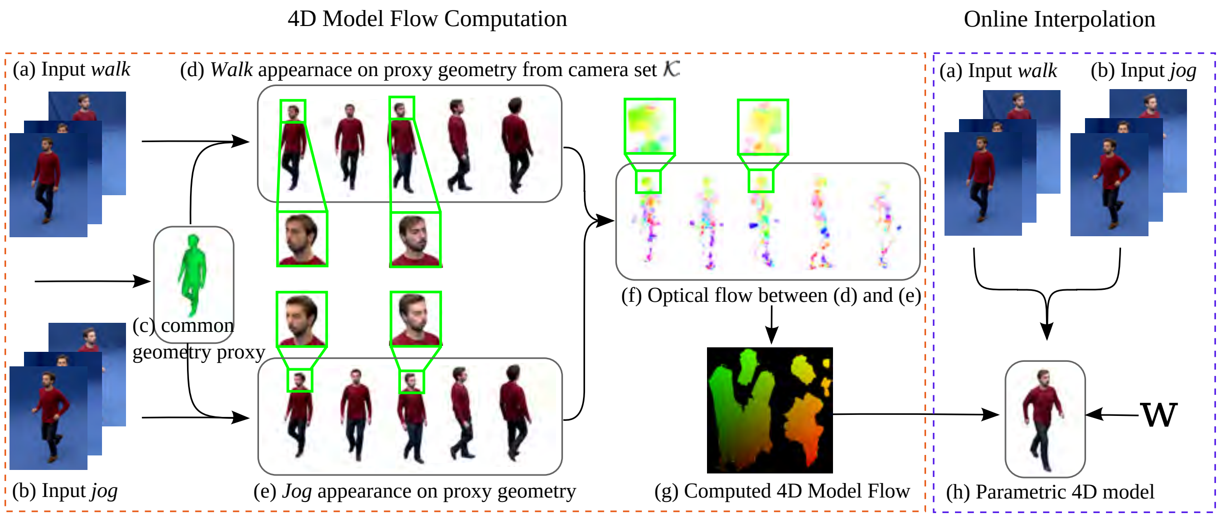
4D Model Flow Computation
Online Interpolation
(a) Input *walk*
(d) *Walk* appearnace on proxy geometry from camera set $\mathcal{K}$
(a) Input *walk*
(b) Input *jog*
(c) common geometry proxy
(f) Optical flow between (d) and (e)
W
(b) Input *jog*
(e) *Jog* appearance on proxy geometry
(g) Computed 4D Model Flow
(h) Parametric 4D model

Figure 3: The full pipeline of our proposed approach, shown here using sample *walk* and *jog* videos: (Left, Section 4) Given two input 4D video models with geometry and multi-view images (a, b), we project the images onto the same proxy geometry (c), and render them from multiple views (d, e), to compute appearance correspondences in each view (f), which we encode as 4D model flow in a UV map on the mesh surface (g). (Right, Section 5) This enables real-time view-dependent video-realistic rendering of a parametric 4D model (h) using the interpolation parameter w.

pearances of the two source 4D video models onto a common 4D model geometry, and then use image-based correspondences in a number of viewpoints to align the appearances (Section 4). The resulting 3D flow vectors are stored efficiently in terms of the mesh's UV domain. The second step, described in Section 5, exploits the precomputed 4D model flows to enable real-time interpolation of 4D video models on the fly.

4. 4D Model Flow for Appearance Alignment

The goal of the 4D model flow is to minimize the visual discrepancy between the appearances of any two 4D models, at all interpolated positions, and for all viewpoints. In this section, we first develop this intuition into a general energy formulation, and then propose an image-based algorithm for estimating dense appearance correspondences. In particular, we exploit a common class of 4D videos with geometric correspondence across 4D models, but with differences in character pose and camera views in each motion [BHKH13]. Finally, we show how to store the precomputed 4D model flows efficiently in 2D UV maps that enable interactive control of both motion and viewpoint in real time (see Section 5).

4.1. Definition of 4D Model Flow

4D performance capture techniques reconstruct 4D videos that are sequences of 4D models $V(t) = (M(t), I(t))$ comprising the 3D mesh $M(t)$, with mesh vertices corresponding to the same geometric surface points over time [BHKH13], and the set of video frames $I(t) = \{I^c(t)\}_{c \in C}$ captured from a set C of cameras [dAST*08]; both meshes and video frames vary over time t. In the following, we consider a generic image-based rendering process $IBR(\mathbf{X}, \pi, M, I)$ [DYB98] that computes the color of a 3D point $\mathbf{X}$ from the viewpoint determined by the projection matrix π, using the mesh M and set of video frames I of a given 4D model $V = (M, I)$.

We define the 4D model flow to be the flow that minimizes appearance differences when interpolating between two 4D models $V_1 = (M_1, I_1)$ and $V_2 = (M_2, I_2)$ on a common geometry proxy $\widehat{M}$. This proxy geometry can in principle be freely chosen, as long as it maintains per-vertex geometric correspondence to both input meshes (M_1 and M_2), and it therefore acts as a common reference frame. However, even with perfect geometric correspondences between the 4D models, their appearance (I_1 and I_2) need not agree when projected onto the same geometry proxy $\widehat{M}$, because of dynamic appearance effects like moving clothes, wrinkles and shading differences. This is visible in the highlighted close-up views in Figure 3 (d and e).

The 4D model flow seeks to minimize this appearance discrepancy using a flow field F that aligns appearances on the surface of the common proxy geometry $\widehat{M}$ (using the forward and backward flows $F_{1,2}$ and $F_{2,1}$, respectively). Specifically, the 4D model flow F is the minimizer of

$$\int_\pi \int_{\mathbf{X}} \max_{w \in [0,1]} \Big\| IBR\Big(\mathbf{X} + w \cdot F_{1,2}(\mathbf{X}), \pi, \widehat{M}, I_1\Big) - IBR\Big(\mathbf{X} + (1-w) F_{2,1}(\mathbf{X}), \pi, \widehat{M}, I_2\Big) \Big\|_2, \quad (1)$$

which minimizes the maximum visual discrepancy between the rendered appearances (wrapped in $\|\cdot\|_2$), across the pose interpolation range $w \in [0,1]$, integrated over all visible points $\mathbf{X}$ on the mesh $\widehat{M}$, and over projection matrices π. In this most general form, this expression is infeasible to minimize exactly because of the highly non-linear nature of the involved image-based rendering process. However, by making a few practical assumptions – such as restricting the optimization to a small number of canonical viewpoints instead of all possible viewpoints – we can simplify this problem and estimate the 4D model flow with an image-based approach, as explained in the next section.

4.2. Image-Based Estimation of 4D Model Flow

Free-viewpoint video rendering maintains the visual realism of the captured videos, but is limited to playback of recorded 4D models [BBM*01]. 4D video textures lift this limitation by interpolating both the geometries and the textures of a set of captured 4D video sequences, allowing the synthesis of parametric 4D videos. However, this requires online texture alignment from the camera viewpoint, which results in a significant computational overhead at run time. To overcome this limitation, we present a novel approach for 4D model flow estimation and storage.

Starting from two input 4D video sequences, $V_1(t)$ and $V_2(t)$, where $V_i(t) = (M_i(t), I_i(t))$ as before, we aim to find and store dense appearance correspondences for each pair of 4D models $\{V_1(t), V_2(t)\}$. However, solving this correspondence problem directly on the input images is non-trivial due to significant differences in the captured pose and appearance. As an example, consider the images in Figure 3 (*a*, *b*), and note the different positions of the right leg. Standard optical-flow-based methods fail to find the correct correspondences in such a challenging scenario.

To solve this problem, we exploit the known per-vertex geometric correspondence across 4D models [BHKH13] by projecting the different input textures onto a common geometry proxy, which facilitates the solution of the correspondence problem. For simplicity, we use the geometry of the first 4D video model as proxy geometry, i.e. $\widehat{M} = M_1(t)$, because it ensures perfect appearance reproduction of the first 4D model when interpolating between the models. As we compute the 4D model flow independently for each time step t, we simplify our notation by making the time argument implicit from now on, and thus simply write V_i for $V_i(t)$, for example.

We project the images I_1 and I_2, extracted from different multi-view videos, onto the proxy geometry $\widehat{M}$ using the image-based renderer *IBR* introduced in the previous section (using Volino et al.'s technique [VCCH14]), to produce the corresponding rendered images $R_i(\pi)$ for the view given by the projection matrix π. The synthesized rendered images result in matching poses, as well as silhouettes, for any given view π, in contrast to the original silhouettes in the input images I_1 and I_2. We then generate a set of renderings $\mathcal{R}_i = \{R_i(\pi_k)\}_{k\in\mathcal{K}}$ for a canonical set of viewpoints $\mathcal{K}$ (5–10 in this paper, see Table 1). These viewpoints can be chosen freely, although equidistant viewpoints along a circle tend to provide a uniform evaluation for rendering. In our experience, the original camera positions of one of the input multi-view sequences work well, and we used them for all results in this paper. This process is depicted in Figure 3 (*a*–*e*), showing the original images for the *walking* and *jogging* sequences (notice the difference in silhouettes in *a* and *b*), in contrast to the reprojected appearance (*d* and *e*), where both sequences share the same silhouettes.

We use the pairs of corresponding camera renderings, $R_1(\pi_k)$ and $R_2(\pi_k)$, for each viewpoint $k \in \mathcal{K}$, to compute the 2D projections of the 4D model flow in image space using off-the-shelf optical flow [Far03]. The resulting flow fields $\{O_k\}_{k\in\mathcal{K}}$ contain dense appearance correspondences across the 4D models and can already be used for texture alignment. However, they are specific for each of the viewpoints in $\mathcal{K}$ (see Figure 3f). This viewpoint dependency forced previous work to recompute the flow fields at run time for each requested viewpoint. In contrast, we parametrize the view-dependent flow fields directly on the mesh surface in the next section. This is more efficient, but most importantly it makes the 4D model flow independent of the viewpoint used at run time.

4.3. Efficient Storage of 4D Model Flow

Parametrizing the 4D model flow on the 2D mesh surface eliminates the viewpoint dependence and allows for efficient storage using a single texture on graphics hardware. Here, we exploit the known surface parametrization (UV map) of our 4D models to convert and store the computed flow fields $\{O_k\}_{k\in\mathcal{K}}$ as follows (see also the outline in Figure 4).

We first define a "base UV map" B that maps pixels to themselves for valid UV coordinates, and to $(0,0)$ otherwise:

$$B(\mathbf{a}) = \begin{cases} \mathbf{a} & \text{if } \mathbf{a} \text{ is a valid UV coordinate for } \widehat{M}, \\ (0,0) & \text{otherwise,} \end{cases} \tag{2}$$

for $\mathbf{a} \in [0,1]^2$. Figure 5a shows a visualization of the base UV map for the character used in Figures 1 and 3, using the red and green color channels to encode the u and v components, respectively.

We then define a "flow UV map" F that will encode the correspondences currently stored in the flows $\{O_k\}_{k\in\mathcal{K}}$. Recall that each flow field O_k contains the correspondences between the rendered images $R_1(\pi_k)$ and $R_2(\pi_k)$ as 2D vectors observed from viewpoint π_k such that $O_k(\mathbf{x}) = \Delta\mathbf{x}$ is the flow vector of the pixel $\mathbf{x}$ in $R_1(\pi_k)$. For each pixel $\mathbf{a}$ in the flow UV map, we first obtain its corresponding 3D point $\mathbf{P}$ on the mesh $\widehat{M}$ to identify the camera viewpoint in $\mathcal{K}$ with the optimal, most direct view using

$$\hat{k} = \underset{k\in\mathcal{K}}{\operatorname{argmin}}\, N(\mathbf{P}) \cdot L_k, \tag{3}$$

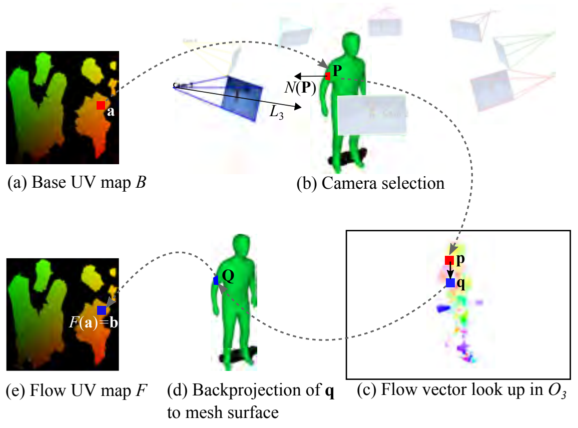

Figure 4: The main steps of encoding the 4D model flow in the flow UV map F. See Section 4.3 for details.

where $N(\mathbf{P})$ is the unit surface normal at $\mathbf{P}$, and L_k the unit look vector of camera k; these directions should be near-opposite for the optimal camera, and hence result in a large negative dot product. We then project the point $\mathbf{P}$ to the screen-space position $\mathbf{p} = \pi_{\hat{k}}\mathbf{P}$ in the optimal view, and look up its flow vector $\Delta\mathbf{p} = O_{\hat{k}}(\mathbf{p})$. We parametrize the screen-space flow $\Delta\mathbf{p}$ on the mesh surface $\widehat{M}$ by first backprojecting the point $\mathbf{q} = \mathbf{p} + \Delta\mathbf{p}$ onto the mesh surface to a point $\mathbf{Q}$, and then storing its UV coordinates $\mathbf{b}$ in the flow UV map F:

$$F(\mathbf{a}) = \begin{cases} \mathbf{b} & \text{if } \mathbf{a} \text{ is a valid UV coordinate for } \widehat{M}, \\ (0,0) & \text{otherwise.} \end{cases} \quad (4)$$

The synthesized flow UV map F encapsulates the 4D model flow, i.e. the appearance correspondences between the input images I_1 and I_2 for all surface points of $\widehat{M}$. See Figure 5b for an example. In practice, we render the base UV map B on the mesh $\widehat{M}$ from viewpoint $\pi_{\hat{k}}$ to easily look up the UV coordinate $\mathbf{b}$ corresponding to the image pixel position $\mathbf{q}$, without computing the point $\mathbf{Q}$.

Figures 5a and 5b illustrate the base and flow UV maps generated from Equations 2 and 4, respectively. The difference between the base and flow UV maps is visualized in Figure 5c, with a brighter version at the bottom to reveal more flow detail. Notice how some border areas in Figure 5b present significant changes in color compared to Figure 5a. This occurs where UV pixel correspondences move across a texture seam. Such across-patch mapping usually appears in non-rigid areas divided by texture seams, which are necessary to parametrize a 3D model in a 2D domain. However, as described in detail in the next section, this does not cause visible artifacts in the final rendering because UV correspondences are projected back into the 3D domain at run time, where flow vectors do not suffer from discontinuous mapping.

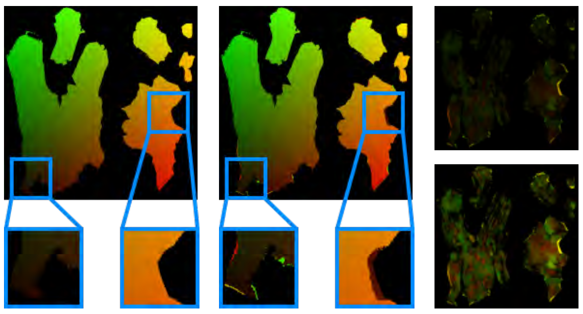

Figure 5: Visualization of the UV maps encoding the dense appearance correspondences across two 4D models: (a) the base UV map B, and (b) the flow UV map F. For better visualization, the relative flow (c) is repeated with increased brightness ($\times 8$) below. The red and green color channels encode the UV components, respectively. Note the wrap-around at patch boundaries in (b), at which flow vectors map across texture seams in the UV map.

5. Real-time 4D Video Interpolation

The 4D model flow computed in the previous section encodes the surface correspondences that minimize appearance differences when interpolating between two 4D models. In this section, we describe how we can use this precomputed 4D model flow to enable view-independent appearance alignment for parametric 4D models in real time, more than an order of magnitude faster than previous work yet with similar visual quality. Without such alignment, any residual misalignment in the underlying geometric correspondences, or any appearance deformation that cannot be reproduced temporally coherently by geometry only (i.e. wrinkles and hair), produces blurring and ghosting artifacts, as visible in Figures 1b and 7a.

5.1. 4D Model Flow for 4D Video Interpolation

Given two 4D video frames V_1 and V_2, where $V_i = (M_i, I_i)$ as before, and a user-specified interpolation weight w, we first compute a parametric geometry proxy $\widehat{M}_w$ by interpolating the meshes M_1 and M_2. We use a non-linear mesh blending approach that interpolates the rotation and change in shape independently per each triangle of the mesh according to the weight w, and performs a least-squares solution to obtain the resulting mesh $\widehat{M}_w$ [CTGH13]. Our aim is to synthesize a parametric texture for $\widehat{M}_w$ that not only maintains the visual fidelity of the acquired images but also changes dynamically to fit to the current pose, by exploiting the precomputed flow UV map F, and the input images I_1 and I_2.

We start by projecting the images I_1 and I_2 onto the proxy geometry $\widehat{M}_w$ using the image-based renderer IBR, to generate images $R_i(\pi)$ using the current viewpoint's projection

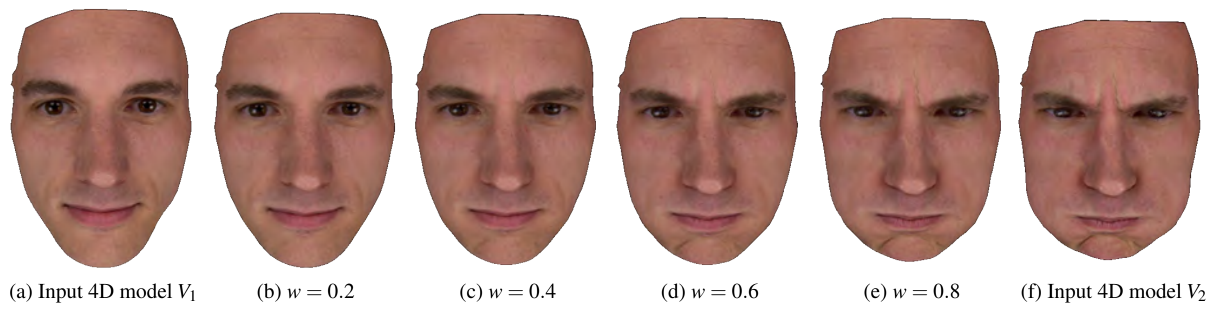

Figure 6: Interpolation between two input expressions (*a*) and (*e*). Intermediate parametric models (*b*–*e*) synthesized using our 4D model flow for dense appearance correspondence between inputs. Dataset from Klaudiny et al. [KBH12].

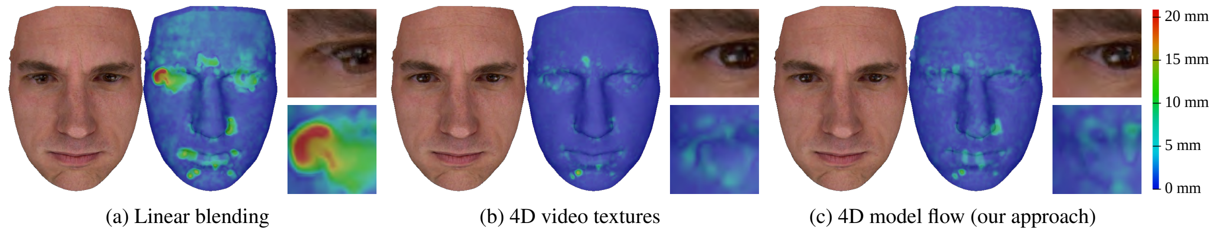

Figure 7: Evaluation of the interpolation in Figure 6, for half-way interpolation ($w = 0.5$). (*a*) Linear blending of appearances results in severe ghosting artifacts. (*b*) 4D video textures mitigate ghosting artifacts by computing appearance alignment in screen space. (*c*) Our proposed approach achieves results comparable to (*b*), without requiring online optical flow computation.

matrix π. The simple linear blending of these renderings, $R_{\text{out}} = (1-w) \cdot R_1(\pi) + w \cdot R_2(\pi)$, results in blurring artifacts due to residual misalignments in the underlying geometry as well as inherent appearance changes in non-rigid surfaces. To correct such ghosting artifacts, we use the precomputed flow UV map F to warp the the rendered frames $R_1(\pi)$ and $R_2(\pi)$ to find the alignment that minimizes visual discrepancies (Equation 1), as follows.

Using the same notation as in Section 4.3, we convert each pixel position $\mathbf{a}$ in the flow UV map F, and its value $\mathbf{b} = F(\mathbf{a})$, to points $\mathbf{P}$ and $\mathbf{Q}$ on the mesh $\widehat{M}_w$, as in Section 4.3. We then scale their difference vector by the interpolation weight w to obtain displacement vectors in world space, $\mathbf{J} = w \cdot (\mathbf{Q} - \mathbf{P})$, and in screen space, $\mathbf{j} = w \cdot (\mathbf{q} - \mathbf{p})$, where $\mathbf{p} = \pi\mathbf{P}$ and $\mathbf{q} = \pi\mathbf{Q}$ are the projections of $\mathbf{P}$ and $\mathbf{Q}$ in the rendered view. We use the screen-space displacement vector $\mathbf{j}$ to warp the pixel $\mathbf{p}$ in $R_1(\pi)$ to $\mathbf{p}' = \mathbf{p} + \mathbf{j}$. Analogously, we obtain the screen-space vector $\mathbf{k} = (1-w) \cdot (\mathbf{p} - \mathbf{q})$ to warp the pixel $\mathbf{q}$ in $R_2(\pi)$ to $\mathbf{q}' = \mathbf{q} + \mathbf{k}$. The final interpolated appearance is obtained by linearly blending between the warped renderings $U_i(\pi)$ obtained from $R_i(\pi)$:

$$R_{\text{out}} = (1-w) \cdot U_1(\pi) + w \cdot U_2(\pi). \tag{5}$$

5.2. Implementation Details

Our approach for appearance alignment is based on online interpolation of corresponding 3D surface points that originate as UV coordinates. Specifically, we need to compute the world-space point $\mathbf{Q}$ from the UV coordinate $\mathbf{b}$ within a pixel shader – a feature that is currently unavailable in modern graphics pipelines.

We therefore use the following approach: in a preprocess, we first render a triangle ID buffer of the mesh in its UV domain. This allows us to easily identify the triangle that contains the UV coordinate $\mathbf{b}$ by looking up $\mathbf{b}$'s location in the ID buffer. The pixel shader then computes the 3D position of $\mathbf{Q}$ by barycentric interpolation from the triangle's vertices, which are looked up by the shader in a texture buffer encoding all triangles in the mesh. This approach computes $\mathbf{Q}$ in a single rendering pass, even when the world-space positions of triangle vertices change during an animation.

6. Results

We implemented our approach using OpenGL 4.3 and C++11. We store each 4D model flow as a 512×512 texture, which provides sufficient resolution for all results shown in this paper, while only requiring 1 MB of memory. The UV mesh parametrization of each character is found by automatically unwrapping the 3D mesh given a set of manually provided surface seams. Note that the quality of our results is not affected by how the UV parametrization is obtained because stored correspondences are projected to the 3D space before appearance warping and blending.

We applied our appearance alignment approach to a variety of publicly available 4D video datasets. Figures 1, 2 and 9 show interpolation results for the full-body models of Casas et al. [CTGH13], and Figures 6 to 8 show interpolation results for the face models of Klaudiny et al. [KBH12]. Please refer to our supplementary video to see animations of our results as well as additional results. Figures 2, 6, 8 and 9 show multiple interpolation results using our approach. All datasets have an image resolution of 1920×1080, but use different numbers of cameras, as listed in Table 1. The output resolution is a 1280×720 animation. We disable texture filtering for lookups in the UV flow map to avoid mixing discontinuous values, which comes at the cost of small inaccuracies caused by the nearest-neighbor sampling, when required.

6.1. Performance

We compare the results of our approach qualitatively and quantitatively against a simple linear blending approach, and the state of the art in dynamic appearance interpolation for 4D models [CVCH14]. Our results achieve similar visual quality but with real-time rendering speeds close to the simple linear blending, which is why we compare against these approaches. We show visual comparisons between these approaches in Figures 1, 7 and 8. Our 4D model flow approach mitigates the blending artifacts of the linear approach (see for example Figures 1b and 7a), and generates results comparable to online screen-space alignment.

	Approach	Figure 1	Figure 6	Figure 8	Figure 9
Average Alignment Error (mm)	Linear	18.7	8.7	6.5	15.5
	4DVT	1.2	0.5	0.6	1.7
	Ours	1.5	0.7	0.6	2.2
Maximum Alignment Error (mm)	Linear	22.3	18.7	15.6	28.7
	4DVT	5.2	3.2	4.5	3.2
	Ours	4.8	4.1	5.3	3.8
Time (s)	Linear	0.003	0.008	0.007	0.003
	4DVT	0.093	0.125	0.152	0.072
	Ours	0.007	0.015	0.015	0.009
Number of cameras		8	5	5	10
Number of vertices		2667	2689	2689	4052

Table 1: Quantitative evaluation of our proposed approach compared to linear texture blending and 4D video textures (4DVT) for the figures presented in this paper.

Table 1 presents a quantitative evaluation of performance in terms of alignment error and run time, for a number of datasets. We compute the alignment error using the magnitude of the optical flow vectors between the warped appearances for $w \in \{0.0, 0.1, \ldots, 1.0\}$, as observed from the viewpoint used in each figure. For perfect alignment, the flow magnitude is zero. We also visualize the alignment error as heat maps in Figures 1, 7 and 8. Our approach reduces the

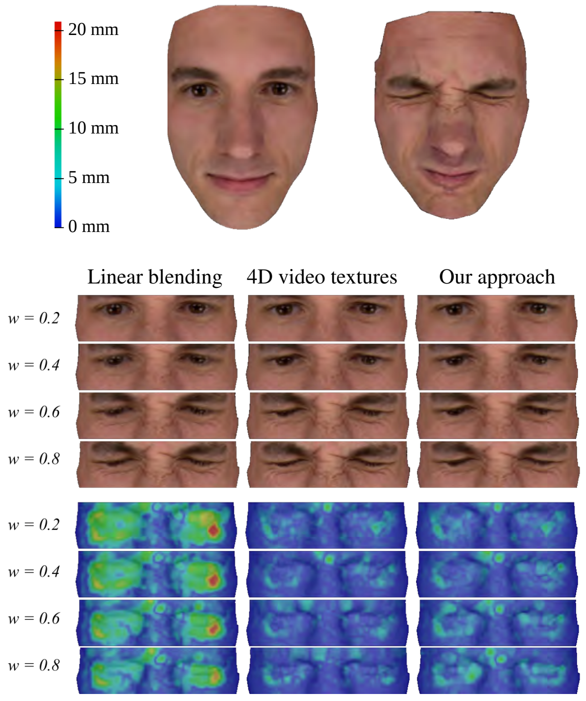

Figure 8: Close-up of the interpolation between the two face 4D models at the top, for multiple interpolation weights w.

rendering costs compared to 4D video textures by an order of magnitude while maintaining similar visual quality.

The precomputation of each 4D model flow (i.e. one time step of a 4D animation) takes about 1 minute for a resolution of 512×512 pixels on a 3 GHz dual-core processor with 8 GB memory and an NVIDIA GeForce GT 640M GPU.

6.2. Discussion

Our appearance alignment approach uses the same optical flow algorithm [Far03] as 4D video textures, but we apply it to the canonical viewpoints and not the rendered viewpoints. This results in a small increase in alignment errors in our approach due to the requirement for reprojecting the flows from the rendered viewpoint. However, this quality gap could be closed in different ways. The 4D model flow computation could be improved with recent further enhanced optical flow techniques [SRB14], which may increase the precomputation time, but the rendering times would be unchanged. One could also improve the multi-view camera coverage of the 4D videos using additional canonical viewpoints or by optimizing the locations of the existing canonical viewpoints. Lastly, one could refine the geometric proxy for each 4D model flow to reduce occluded regions.

Similarly, the selection of which camera is chosen to evaluate the alignment for a given mesh fragment could also be

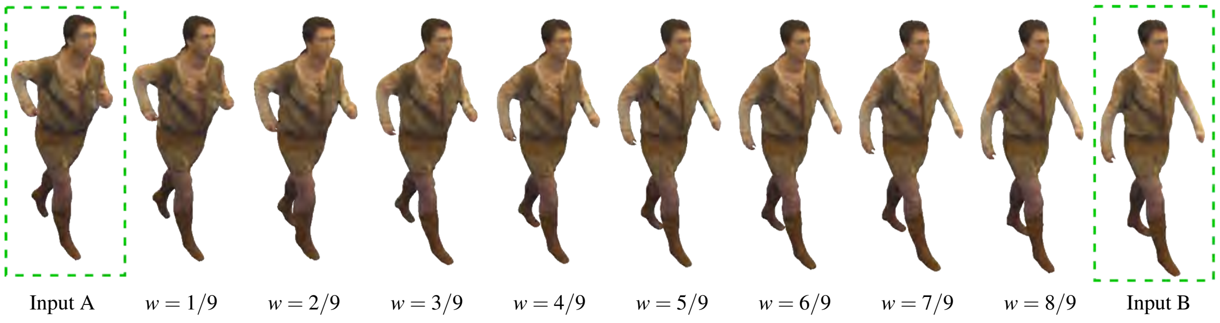

Figure 9: Interpolation results between two 4D video models (left and right) from different sequences.

improved in the future. We currently use the most direct camera to each fragment, however, this can lead to a fragmented assignment map. Optimizing such camera assignment could provide a more uniform flow UV map.

An important limitation of the proposed approach is the need for a flow UV map for each pair of input models. Each computed flow UV map requires 1 MB of memory to store a two-component 16-bit image with 512×512 pixels. However, it is important to remark that we do not need a 4D model flow for all possible pairs of frames in a dataset, only between the frames which we want to interpolate. For example, the walk to jog motion presented in the supplementary video only requires 28 flow maps to be computed. Animations used for Figures 2a and 2b required 26 and 32 flow maps, respectively.

We believe that the 4D model flow can help in better evaluating how the space-time appearance changes in multi-camera captured datasets. Current approaches for surface tracking techniques that rely on texture cues could benefit from our proposed representation.

7. Conclusion

4D model flow enables the precomputation and storage of dense correspondences across a multi-view dataset, which is shown to allow fast accurate appearance alignment at run time. Our results demonstrate that aligned 4D models exploiting the proposed representation achieve equivalent results to previous work for appearance alignment, with an order of magnitude reduction in computational cost at render time. This allows real-time rendering of interpolated 4D video sequences.

As our results show, the 4D model flow clearly and consistently improves the visual quality of renderings of parametric 4D models over purely geometry-driven blended textures ("linear blending"), for any given quality of geometric correspondences, irrespective the mesh resolution. By precomputing 4D model flows and storing them compactly, they can be used for real-time rendering of 4D video textures for parametrically controlled surface animation.

Acknowledgment. This research was funded by the ERC Starting Grant project CapReal (335545) and the InnovateUK project REFRAME.

References

[BBM*01] Buehler C., Bosse M., McMillan L., Gortler S., Cohen M.: Unstructured lumigraph rendering. In *SIGGRAPH* (2001), pp. 425–432. 1, 5

[BHB*11] Beeler T., Hahn F., Bradley D., Bickel B., Beardsley P., Gotsman C., Sumner R. W., Gross M.: High-quality passive facial performance capture using anchor frames. *ACM Transactions on Graphics (SIGGRAPH) 30*, 4 (2011), 75:1–10. 1, 2

[BHKH13] Budd C., Huang P., Klaudiny M., Hilton A.: Global non-rigid alignment of surface sequences. *International Journal of Computer Vision 102*, 1–3 (2013), 256–270. 1, 2, 4, 5

[BHPS10] Bradley D., Heidrich W., Popa T., Sheffer A.: High resolution passive facial performance capture. *ACM Transactions on Graphics (SIGGRAPH) 29*, 4 (2010), 41. 2

[CTGH13] Casas D., Tejera M., Guillemaut J., Hilton A.: Interactive animation of 4D performance capture. *IEEE Transactions on Visualization and Computer Graphics 19*, 5 (2013), 762–773. 2, 3, 6, 8

[CVCH14] Casas D., Volino M., Collomosse J., Hilton A.: 4D video textures for interactive character appearance. *Computer Graphics Forum (Eurographics) 33*, 2 (2014), 371–380. 1, 2, 3, 8

[dAST*08] de Aguiar E., Stoll C., Theobalt C., Ahmed N., Seidel H.-P., Thrun S.: Performance capture from sparse multi-view video. *ACM Transactions on Graphics (SIGGRAPH) 27*, 3 (2008), 98:1–10. 1, 2, 4

[DYB98] Debevec P., Yu Y., Borshukov G.: Efficient view-dependent image-based rendering with projective texture-mapping. In *Rendering Techniques* (1998), pp. 105–116. 1, 4

[EDDM*08] Eisemann M., De Decker B., Magnor M., Bekaert P., de Aguiar E., Ahmed N., Theobalt C., Sellent A.: Floating textures. *Computer Graphics Forum (Eurographics) 27*, 2 (2008), 409–418. 2

[Far03] Farnebäck G.: Two-frame motion estimation based on polynomial expansion. In *Image Analysis (Proceedings of SCIA)* (2003), pp. 363–370. 5, 8

[FNZ*09] Flagg M., Nakazawa A., Zhang Q., Kang S. B., Ryu Y. K., Essa I., Rehg J. M.: Human video textures. In *I3D* (2009), pp. 199–206. 2

[HHS09] HUANG P., HILTON A., STARCK J.: Human motion synthesis from 3D video. In *CVPR* (2009), pp. 1478–1485. 2

[KBH12] KLAUDINY M., BUDD C., HILTON A.: Towards optimal non-rigid surface tracking. In *ECCV* (2012). 7, 8

[LLN*14] LIAO J., LIMA R. S., NEHAB D., HOPPE H., SANDER P. V.: Semi-automated video morphing. *Computer Graphics Forum (EGSR) 33*, 4 (2014), 51–60. 2

[LYT11] LIU C., YUEN J., TORRALBA A.: SIFT flow: Dense correspondence across scenes and its applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*, 5 (2011), 978–994. 2

[PKF07] PONS J.-P., KERIVEN R., FAUGERAS O.: Multi-view stereo reconstruction and scene flow estimation with a global image-based matching score. *International Journal of Computer Vision 72*, 2 (2007), 179–193. 3

[QBDC14] QUIROGA J., BROX T., DEVERNAY F., CROWLEY J.: Dense semi-rigid scene flow estimation from RGBD images. In *ECCV* (2014), pp. 567–582. 3

[SH07] STARCK J., HILTON A.: Surface capture for performance-based animation. *IEEE Computer Graphics and Application 27* (2007), 21–31. 2

[SRB14] SUN D., ROTH S., BLACK M. J.: A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision 106*, 2 (2014), 115–137. 2, 8

[TFB14] TSIMINAKI V., FRANCO J.-S., BOYER E.: High resolution 3D shape texture from multiple videos. In *CVPR* (2014), pp. 1502–1509. 1, 2

[VBMP08] VLASIC D., BARAN I., MATUSIK W., POPOVIĆ J.: Articulated mesh animation from multi-view silhouettes. *ACM Transactions on Graphics (SIGGRAPH) 27*, 3 (2008), 97:1–9. 2

[VBR*05] VEDULA S., BAKER S., RANDER P., COLLINS R., KANADE T.: Three-dimensional scene flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 3 (2005), 475–480. 2, 3

[VBZ*10] VALGAERTS L., BRUHN A., ZIMMER H., WEICKERT J., STOLL C., THEOBALT C.: Joint estimation of motion, structure and geometry from stereo sequences. In *ECCV* (2010), pp. 568–581. 3

[VCCH14] VOLINO M., CASAS D., COLLOMOSSE J., HILTON A.: Optimal representation of multiple view video. In *BMVC* (2014). 3, 5

[VSR13] VOGEL C., SCHINDLER K., ROTH S.: Piecewise rigid scene flow. In *ICCV* (2013), pp. 1377–1384. 3

[WBV*11] WEDEL A., BROX T., VAUDREY T., RABE C., FRANKE U., CREMERS D.: Stereoscopic scene flow computation for 3d motion understanding. *International Journal of Computer Vision 95*, 1 (2011), 29–51. 3

[XLS*11] XU F., LIU Y., STOLL C., TOMPKIN J., BHARAJ G., DAI Q., SEIDEL H.-P., KAUTZ J., THEOBALT C.: Video-based characters: Creating new human performances from a multi-view video database. *ACM Transactions on Graphics (SIGGRAPH) 30*, 4 (2011), 32:1–10. 2

[ZBH12] ZAHARESCU A., BOYER E., HORAUD R.: Keypoints and local descriptors of scalar functions on 2D manifolds. *International Journal of Computer Vision 100* (2012), 78–98. 2

[ZK14] ZHOU Q.-Y., KOLTUN V.: Color map optimization for 3D reconstruction with consumer depth cameras. *ACM Transactions on Graphics (SIGGRAPH) 33*, 4 (2014), 155:1–10. 2

[ZKU*04] ZITNICK C. L., KANG S. B., UYTTENDAELE M., WINDER S., SZELISKI R.: High-quality video view interpolation using a layered representation. *ACM Transactions on Graphics (SIGGRAPH) 23*, 3 (2004), 600–608. 2