Layered View-Dependent Texture Maps

Marco Volino and Adrian Hilton
Centre for Vision, Speech and Signal Processing
University of Surrey, Guildford, GU2 7XH, UK
{m.volino,a.hilton}@surrey.ac.uk

ABSTRACT

Video-based free-viewpoint rendering from multiple view video capture has achieved video-realistic performance replay. Existing free-viewpoint rendering approaches require storage, streaming and re-sampling of multiple videos, which requires high bandwidth and computational resources limiting applications to local replay on high-performance computers. This paper introduces a layered texture representation for efficient storage and view-dependent rendering from multiple view video capture whilst maintaining the video-realism. Layered textures re-sample the captured video according to the surface visibility. Prioritisation of layers according to surface visibility allows the N-best views for all surface elements to be pre-computed significantly reducing both storage and rendering cost. Typically 3 texture map layers are required for free-viewpoint rendering with an equivalent visual quality to the multiple view video giving a significant reduction in storage cost. Quantitative evaluation demonstrates that the layered representation achieves a 90% reduction in storage cost and 50% reduction in rendering cost without loss of visual quality compared to storing only the foreground of the original multiple view video. This reduces the storage and transmission cost for free-viewpoint video rendering from eight cameras to be similar to the requirements for a single video. Streaming the layered representation enables, for the first time, demonstration of free-viewpoint video rendering on mobile devices and web platforms.

Categories and Subject Descriptors

I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*

Keywords

3D Video, Video-based Rendering, Texture Mapping

1. INTRODUCTION

In recent years, the use of video-based rendering from multiple-view capture and reconstruction has become a popular method for producing three dimensional (3D) content due to the high-level of realism. The capture of human performance in particular has a wide range of applications including the production of realistic characters for film, broadcast and gaming. Traditional real-time rendering engines (e.g. gaming engines) attach a single texture map to the surface of the model once for all frames. While this is efficient in terms of storage, the temporal dynamics of the surface are lost, such as creases in clothing and wrinkles in the skin. Free-viewpoint video rendering, also known as view-dependent rendering, produces a novel view of a scene from a user-controlled viewpoint. The rendered appearance is sampled directly from the captured frames, which preserves the realism of the captured video by reproducing surface dynamics and the effects of non-Lambertian reflectance.

A major issue with the video-based rendering from multi-view capture is the large storage footprint and bandwidth required for video-rate transfer of data from disk to the GPU at render time. A typical one minute, eight camera sequence captured at full HD resolution (1920x1080) at 25 frames per second (FPS) requires approximately 48 GB of storage space. This storage size and the required bandwidth 800 MB/s limit the application of free-viewpoint video to high performance computers. A common storage method is to extract each frame and mask the background leaving only the foreground pixels related to the actor. This can reduce the storage footprint by 90-95% depending on the camera framing. This reduces the storage cost for eight camera views to approximately 2.4 GB/minute and requires a storage-GPU bandwidth of 40 MB/s. This remains prohibitive for mobile and web applications and highlights the need for a compact and efficient multi-view texture representation that preserves view-dependent features of the captured video.

In this paper, we present a novel layered texture map representation and rendering scheme. The layered texture representation re-samples the captured multiple view video into texture map layers prioritised by the surface visibility. Consequently, the first texture map layer stores the appearance information for each surface element from the best camera view with subsequent layers storing next best appearance information. This allows either a single 'best' texture map to be derived for each frame or a set of texture map layers representing the N-best camera views for use in view-dependent rendering. This provides an efficient representation for video-based rendering from multiple view video which maintains the view-dependent dynamic visual quality whilst reducing the redundancy inherent in the original camera views. The layered representation pre-computes surface visibility, occlusion and projective texturing significantly reducing the rendering cost. The contributions of this paper are as follows:

- A novel layered texture map representation for view-dependent rendering from multi-view video data that decreases the required storage and data transfer by $> 90\%$ whilst maintaining visual quality.
- An efficient free-viewpoint rendering scheme which removes the need for occlusion testing and projective texturing at runtime.

Quantitative comparison of storage cost and rendering quality and cost is performed for free-viewpoint video rendering using the layered representation against a state-of-the-art free-viewpoint video render (FVVR) [13]. This demonstrates that the layered representation achieves a significant reduction in storage and data transfer cost without loss of visual quality in view-dependent rendering. Layered representation is demonstrated to enable mobile and web-based application of free-viewpoint video.

2. BACKGROUND

2.1 Free-Viewpoint Video Rendering

Free-viewpoint rendering (FVVR) creates a novel view of a scene based upon the position of a user-controlled *virtual camera*. FVVR reproduces the changes in the appearance of the surface of the model with the viewing position due to lighting, non-Lambertian surface properties and self-shadowing which maintains the visual realism of the captured video. The ultimate goal of FVVR is to produce novel photo-consistent views, such that the synthesised view would be indistinguishable from a real camera view taken from the same position. Many applications for the technology exist including broadcast (e.g. analysis of sport [8]) and realistic characters for gaming and film. The effect of view-dependence is generally achieved by blending the captured camera views to create the novel view.

The idea of free-view point rendering was first introduced by Chen and Williams [3] who introduced an approach to interpolate between multiple images for synthesised 3D scenes. Debevec *et al.* [5] exploited view-dependent rendering from captured photographs together with a 3D geometric proxy as a way of creating realistic views of static architectural scenes using approximate geometry models. The work was then extended with the aim of making the process more efficient [4] by pre-computing the visibility of each polygon in the capture cameras. The visibility constraints were stored in a data structure known as a 'view map'. The view map assigns each polygon of the mesh a unique ID and associates with the ID a list of cameras in which the polygon are visible. When drawing each polygon, the view map is queried using the position of the virtual camera and returns the three most direct cameras and associated weightings used to blend the views. The texture information is then sampled directly from the captured frames.

Kanade *et al.* [9] introduced the 'virtualized reality' system which captured dynamic human performance from multiple camera views, reconstructed the geometry and allowed the viewer to replay from a desired viewpoint pioneering the first free-viewpoint system. To overcome the limitations of errors in geometry and calibration, Eisemann *et al.* [6] introduced *Floating Textures* which perform optical flow alignment of rendered appearance from multiple views to achieve improved visual quality.

Starck *et al.* [13] released an open-source free-viewpoint video render (FVVR) which is used as a benchmark for the evaluation of the layered free-viewpoint video renderer (LFVVR) presented in this paper. This implementation is based on earlier approaches to free-viewpoint rendering [4] and represents a typical state-of-the-art approach. Given a desired novel view for rendering, FVVR is composed of four steps:

1. **Camera Selection:** A subset of M of the N capture cameras closest to the desired view is chosen to texture the model. The selection is based upon the angle between the virtual camera and the capture camera view directions.
2. **Render Depth Maps:** Depth maps are rendered from the viewpoint of each of the M texturing cameras. The mesh is drawn offset multiple times into the depth buffer. This avoids projective texturing errors at occlusion boundaries caused by approximate geometry.
3. **Calculate Weighting:** The contribution of each of the M texturing camera is calculated using the depth maps to test for self occlusions to remove projection errors.
4. **Blend Textures:** The mesh is projectively textured by projecting each vertex into the camera domain to sample the colour. This is then multiplied by the weighting of the associated camera.

Previous free-viewpoint rendering techniques directly use the original video frames for rendering. Therefore, they have large storage and data transfer overheads which limit their use to high-performance local rendering platforms and prohibit their use on mobile or web-based applications. The approach presented in this paper also pre-computes mesh visibility and projective texturing by re-sampling the texture information into a parametrised UV texture domain. This reduces the rendering of novel views to an efficient texture look-up.

2.2 Texture Maps from Multiple Views

A number of approaches have been proposed to obtain a single texture map from multiple camera views of an object. A key problem is selecting of the optimal blending between cameras. The use of too many cameras leads to blurring of the texture if the geometry or camera calibration is not exact, and the use of too few cameras leads to visible seams between camera boundaries.

Starck *et al.* [11] recover a multi-view texture map by sampling the texture on a per polygon basis from the camera whose view was most orthogonal to the polygon. This maximises the texture resolution in the final image. To reduce the visibility of the seams in the final texture map, areas of overlap are blended. Ziegler *et al.* [16] presented a method where the texture of a human actor was averaged over all views to produce a texture map for each frame in a sequence. A base layer was produced by averaging all texture maps in the sequence. The texture map sequence was encoded using an MPEG video codec that achieved a compression ratio of 50:1 compared to the original frames.

A number of approaches have been introduced to optimise the sampling from multiple view video. Lempitsky and Ivanov [10] use a Markov Random Field (MRF) energy optimisation to maximise the quality of each texture fragment. The seams between the texture fragments are minimised using a procedure similar to gradient-domain based stitching. Xu *et al.* [15] used a similar MRF energy optimisation technique as [10]. They then use radiometric correction to adjust the colour difference between texture fragments to reduce the seams at boundaries. Goldelucke *et al.* [7] formulate the problem of re-sampling from multiple views a global super resolution problem optimising the alignment of appearance across all views on the object surface. Their approach is demonstrated to achieve improved resolution for static shape assuming an accurate shape prior and Lambertian reflectance.

In all cases, the multiple view appearance is combined without any knowledge of the desired rendering view-point resulting in loss of any view-dependent reflectance due non-Lambertian reflectance. The use of a single texture map for dynamic scenes also results in a loss of dynamic detail such as cloth wrinkling or facial expression. The layered representation introduced in this work supports efficient view-dependent rendering and maintains dynamic detail to reproduce the video realism of the captured scene in free-viewpoint rendering.

3. LAYERED TEXTURE MAPS

After capturing a multi-view sequence, a 3D mesh is reconstructed for each frame t in a sequence, where $t = 1...T$ and T is the sequence length. The reconstruction pipeline, shown in Figure 1, consists of silhouette extraction via chroma keying to produce a visual hull that is then refined by matching stereo features [12]. This produces a mesh sequence with no correspondence between the number of vertices or vertex connectivity across frames. We perform non-rigid mesh alignment [1] which produces a temporally aligned mesh M_t at each frame in the sequence. The mesh alignment step is important as the number of vertices and vertex connectivity now remain constant, the only property that now changes is the 3D vertex positions. This allows a single set of texture coordinates, t_c, to be defined which remain valid over the entire sequence.

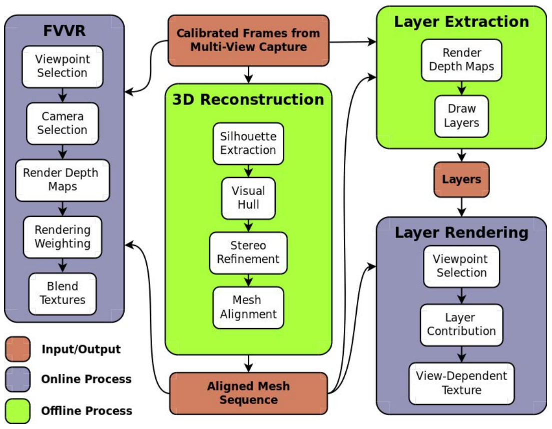


Figure 1: The 3D reconstruction pipeline (centre), the FVVR rendering pipeline (left) and layered extraction and rendering pipeline (right). The colour of the block defines an input/output or online/offline process.

Figure 2 illustrates how M_t is mapped into the 2D texture map domain using t_c. The vertices of M_t can also be projected into the camera domain using π_j, the projection matrix of the j^{th} camera.

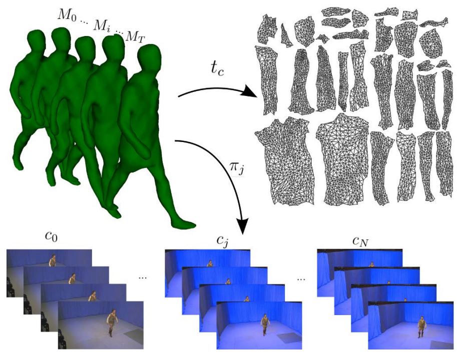


Figure 2: Mapping a mesh sequence into the 2D texture domain using texture coordinates and into the camera domain using a camera projection matrix.

The layered texture representation comprises a set of layers, l_i^t, where $i = 1...L$, where L is the number of layers for each frame, t, in a sequence. A layer is a four channel image that has the colour information stored in the RGB channels and a grey-scale value used to identify the capture camera from which the colour information originated. This grey-scale camera ID is stored in the alpha channel of the image. Layers are ordered such that l_1^t contains the 'best view' for each pixel and subsequent layers contain views that are rated in order of visibility to the capture cameras. This makes the first layer the most complete with reducing completeness for subsequent layers. The layers are generated once offline and stored for use at run-time. The camera assignment information is stored in a single channel grey-scale PNG image. The texture colour can be stored as either images or video. The layer images/videos are loaded at runtime with the corresponding camera assignment map and texture map combined into a single RGBA texture for rendering. The extraction and rendering processes can be efficiently implemented using a single GLSL shader for each process, taking advantage of the speed of the GPU.

This method can also be applied to mesh sequences that have not been aligned provided a set of texture coordinates is defined for each mesh in the sequence. In this paper, only the use of aligned meshes is considered as this allows layer sequences to be efficiently stored using video formats to further reduce the required storage by removing temporal redundancies in the texture information.

3.1 Extraction

The method of extracting the layered texture representation from a multi-view frame is outlined in Figure 1. The extraction process begins by rendering a depth map from each of the capture cameras. The visibility of the vertex is checked in each of the camera views by projecting the vertex into the camera domain and comparing the depth value observed in the depth maps to the distance of the vertex from the camera position. The conditions for visibility testing are shown in Equation 1.

$$D(v, j) = \begin{cases} |v - c_j^P| \leq d & \text{visible} \\ |v - c_j^P| > d & \text{not visible} \end{cases} \qquad (1)$$

where v is the 3D position of a vertex of mesh M_t, c_j^P is the 3D position of the j^{th} camera, and d is the depth observed by projecting v into depth map D_j^t.

In addition to the depth test, the angle between the vertex normal and the camera view direction must lie between zero and 90 degrees. This condition ensures that the vertex is facing the camera. If a vertex is visible in a camera view, it is assigned a score equal to the angle between the vertex normal and camera view direction, as shown in Equation 2. If the vertex is not visible, it is assigned a score of 100 as this is greater than the maximum score of a visible vertex.

$$\theta(v, j) = \cos^{-1}(-c_j^D \cdot n_v) \tag{2}$$

where c_j^D is a unit length view direction vector of the j^{th} camera, n_v is a unit length surface normal vector of vertex v, and θ is the angle between the two vector.

The cameras are then sorted based upon the score from lowest to highest. This orders the cameras in terms of the directness to the vertex normal. The final colour for a fragment in the i^{th} layer is then sampled from the i^{th} camera in the sorted list. This means the first layer is composed using the most direct view of each point on the surface of the mesh with decreasing directness for subsequent layers. The maximum number of layers that can be produced is equal to the number of cameras used to capture the scene. In practice for cameras surrounding the scene, the number of cameras from which most points on the surface are visible will be fewer. For example, in a studio with 8 cameras in a circular configuration most points on the surface are visible from 3-4 views.

The extraction process is summarised as follows:

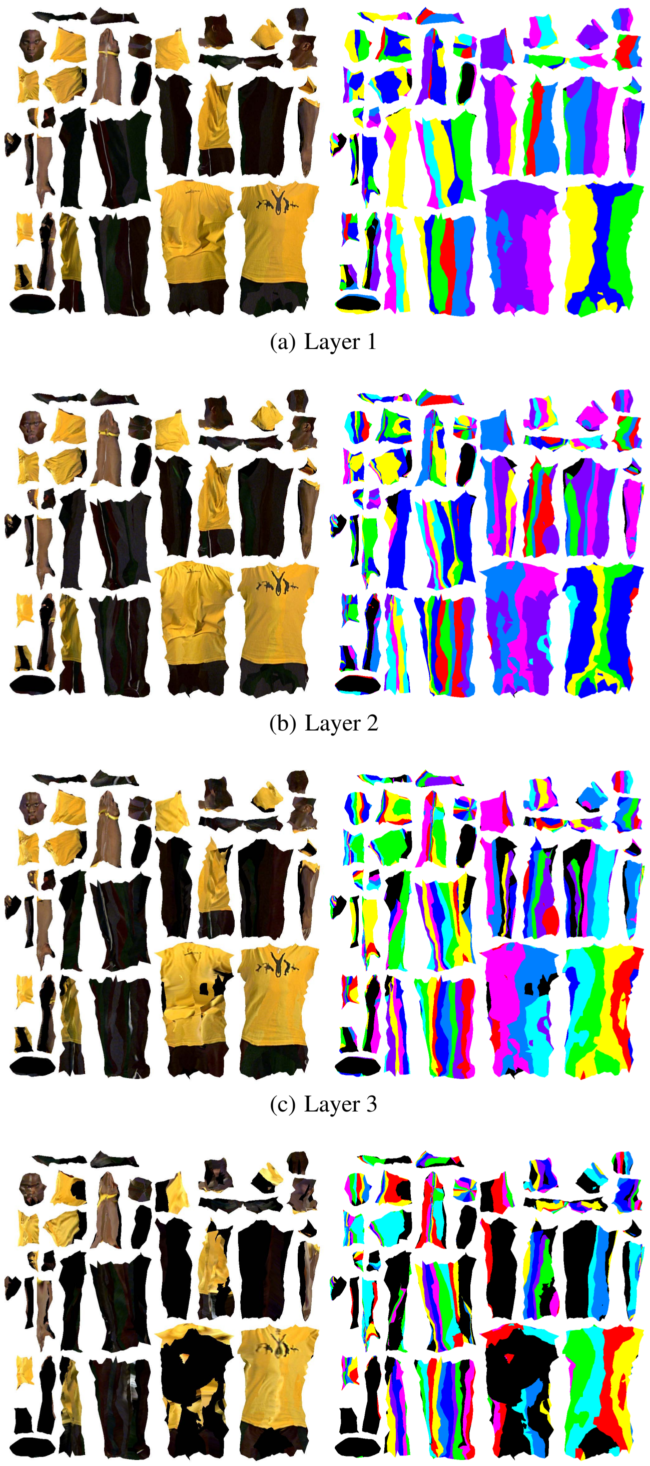
INPUT: Aligned mesh M_t with texture coordinates t_c, calibrated cameras c_j with corresponding projection matrix π_j, capture images I_j^t and depth maps D_j^t

For each pixel in the i^{th} layer of frame t:

1. **Assess Visibility:** The visibility of the associated vertex is assessed in each of the capture cameras by a depth test (Eq.1) and ensuring the angle between the vertex normal and the camera view direction is less than 90 degrees.
2. **Rank Cameras:** Visible cameras are assigned a score (Eq.2), this is equal to the angle between the vertex normal and the camera view direction. Cameras that are not considered to be visible are assigned a default score of 100 as this is greater than the maximum score of a visible camera.
3. **Sort Cameras:** The list of cameras is sorted based upon the assigned score from lowest to highest. This orders the cameras in terms of directness to the vertex normal.
4. **Sample Colour:** The colour is then sampled from the i^{th} camera in the sorted list and stored in the RGB colour channels. In the alpha channel, a grey-scale value is stored which identifies the selected camera. If the vertex is not visible in the i^{th} camera, the pixel is discarded.

OUTPUT: l_i^t, the i^{th} layer of frame t .

An example of four layers from a typical frame are shown in Figure 3. It can be seen that as the layer number increases the layers become more sparse due to occlusions.


(a) Layer 1

(b) Layer 2

(c) Layer 3

(d) Layer 4

Figure 3: Layers 1-4 for a typical frame with each layer split into a texture map (left) and camera assignment map (right). Note: the grey-scale value has been replaced with colours to represent the assignment of different cameras.

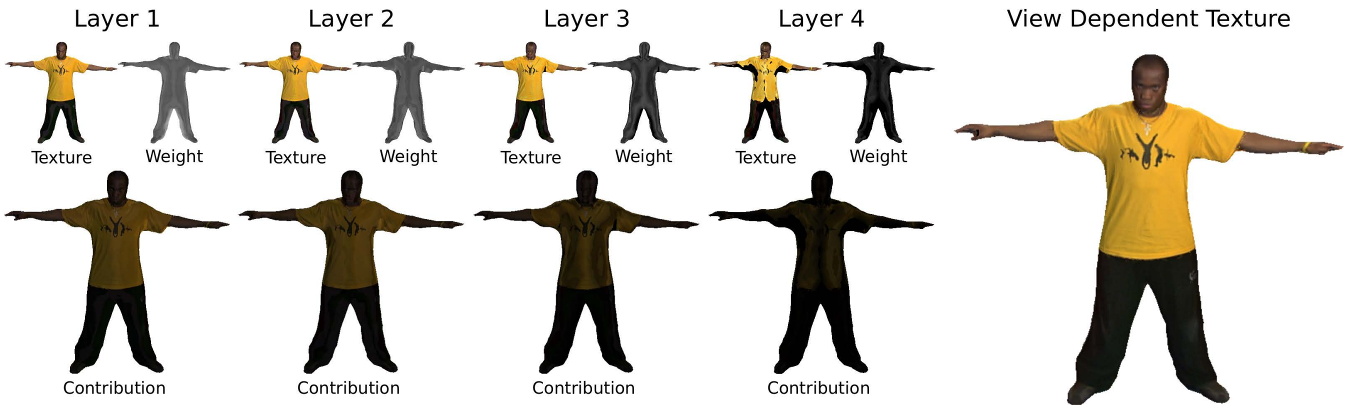


Figure 4: Layered Rendering. Texture, normalised weighting and layer contribution for the first four layers and the final view-dependent texturing of the model. As the layer number is increased, the contribution of the layer becomes less.

3.2 Rendering

The online rendering stage computes the texture of the model based upon the user-selected virtual camera, c_v. When rendering, we make available L layers depending upon the desired quality and available system resources. An overview of the process is shown in Figure 1 and details given below.

For each output fragment of the mesh, we calculate the contribution each layers makes to the final colour based upon the direction of the virtual camera. In order to do this, we first have to identify which camera the pixels in the layers originated from. To identify the cameras, we compare the grey-scale ID, stored in the alpha channel, to the each camera ID passed to the shader program. Once a camera has been identified, the weighting is calculated as shown in Equation 3. If a camera is not able to be identified, the pixel is discarded. The method is independent of the weighting scheme used, however it is required that the weights sum to 1. The weighting scheme that is implemented was chosen to keep it consistent with [13] for comparison purposes.

$$w(v,j) = MAX((c_j^D \cdot n_v)(c_j^D \cdot c_v^D), 0.0) \tag{3}$$

where w is the weighting given a vertex v of mesh M_t and a camera index j, c_j^D is the unit length view direction vector of the j^{th} camera, c_v^D is the view direction of the virtual camera and n_v is the vertex normal of v .

Once weightings have been calculated for all L layers, the weights are normalised with respect to the sum of all the weights. The final view-dependent colour for the output fragment is given by the sum of the RGB colour for each layer scaled by the associated weighting, as shown in Equation 4.

$$f(l^t, c_v) = \sum_{i=1}^{L} w_i RGB_i \tag{4}$$

where f is the fragment colour given a set of layers l^t and a virtual camera position c_v, L is the number of available layers, w_i is the normalised weighting for a fragment of the i^{th} layer, RGB_i is the colour for a fragment of the i^{th} layer.

The rendering process is summarised as follows:

INPUT: Aligned mesh sequence M_t with texture coordinates t_c, calibrated cameras c_j, layers l_i^t where $i = 1...L$, and virtual camera position c_v.

For each output fragment in the mesh:

1. **Calculate contribution of each layer:** Using the grey-scale ID stored in the alpha channel of the layer, the matching camera is identified. Once the camera is known a weighting is calculated (Eq.3). This is based upon the angle between the matched camera view direction, virtual camera view direction and vertex normal. If the ID does not match any of the cameras, the weighting is set to zero.

2. **Calculate view-dependent colour:** The weighting for all L matched cameras is normalised and the final view-dependent colour of the fragment is calculated (Eq.4).

OUTPUT: View-dependent textured mesh

---

4. RESULTS & DISCUSSION

In this section, we present an evaluation of the layered texture map representation and rendering scheme against the open-source FVVR [13]. We compare both approaches in terms of storage, GPU memory requirements, and rendering quality using the FVVR as a benchmark. FVVR is configured to achieve maximum quality by using all available cameras for texturing, M=N. Performance of LFVVR is then evaluated for different numbers of layers L.

The sequences used to evaluate the LFVVR are from the publicly available SurfCap dataset [12]: Roxanne GameCharacter (194 frames), JP (1800 frames). The dataset was captured in a dedicated multi-view studio using eight cameras position uniformly around an 8m diameter circle positioned 2m above the studio floor. This gives a capture volume of approximately 4m x 4m x 2m. The cameras capture frames in 1920x1080 resolution at 25 FPS.

4.1 Rendering Quality

To assess the visual quality, we render the model using both methods from the same viewpoint and compare the output using the Structural Similarity Index Measure (SSIM). SSIM has been chosen as it is more correlated to perceived similarity than other objective measures, such as PSNR [14]. There are two parameters which can affect the rendering quality: the number of available layers, L, and the layer size. We test power-of-two texture sizes from 256 to 2048 as this covers a large range also graphics cards are optimised to deal with power-of-two textures. Figure 5 shows the effect of varying L on the final rendering quality for different layer sizes.

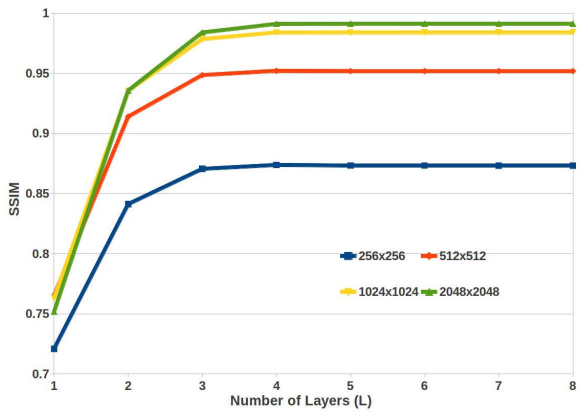


Figure 5: Rendering quality: the effect of varying L on the rendering quality for different values of M. The quality increases up to L=3, after which no significant increase occurs.

It can be seen that as L is increased, the quality of the final render increases with respect to the FVVR for all texture sizes. However, this increase in quality only occurs up to $L = 3$ after which it plateaus. Figure 4 shows that the texture information contained in $L > 3$ contributes very little to the final colour of the rendered model. The amount of texture information in $L > 3$ also decreases. This turning point is due to the set-up of the capture studio as four cameras generally cover half the model.

It can also be seen that increasing the layer size increases the final rendering quality upto 1024. Increasing the layer size further, to 2048, only produces a marginal improvement in quality as this resolution is above that of the captured video. In the smaller layer size (256), the texture is down-sampled from the original image resolution resulting in blurring which can clearly be seen in the heat maps, shown in Figure 6. High frequency details, such as the wrinkles in the clothing and facial features, have been lost and are highlighted as errors. Results of the rendering quality for the Roxanne model are shown in Figure 9 together with heat maps that highlight the differences between the LFVVR rendering and the FVVR benchmark. Further examples for the JP model are shown in Figure 10. These results demonstrate that with $L = 3$ and layer size ≥ 512, the proposed layered texture map representation achieves an equivalent visual quality to state-of-the-art free-viewpoint video rendering.

4.2 Storage

Storage requirements are assessed by comparing the sum of the file size on disk of the layered representation to the camera images with only foreground (masking the background pixels). Table 1 shows the storage cost of two sequences from the evaluation dataset.

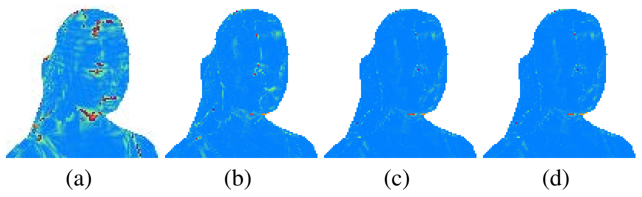


Figure 6: Rendering errors from using different layer sizes. (a) 256, (b) 512, (c) 1024 and (d) 2048

Table 1: Total storage and % reduction for selected sequences in the evaluated datasets

Dataset	JP		Roxanne	
Sequence	Kick-Up		Stand-to-Walk	
Cameras	8		8	
Frames	300		31	
Storage	**(MB)**	**(%)**	**(MB)**	**(%)**
Raw	14238	-	1471	-
Compressed	8116	43	820	44
Masked	422	95	34.5	95

4.2.1 *Image Storage*

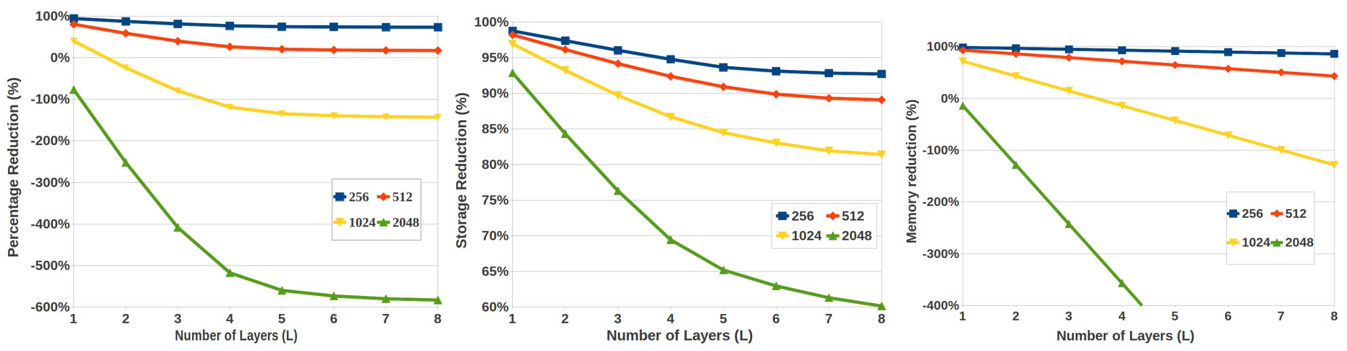
Figure 7(a) shows the reduction in storage achieved for different numbers of layers at different layer sizes when stored using images. This was calculated by comparing the average per frame storage using the layered texture representation with the average per frame storage for all eight frames when the background had been masked (approximately 1.5 MB). As L is increased, the required storage increases resulting in less of a reduction in the required storage. This relationship is true for all layer sizes. As L is increased, the required storage also increases, however the rate at which it increases slows. This is due to the texture information in the layers $L > 4$ becoming sparse. At $L = 3$ with a layer size of 512, the storage reduction is approximately 50% with the layered representation for equivalent rendering quality.

4.2.2 *Video Encoding of Layer Sequences*

The layered texture map representation introduces temporal coherence which is not present in the original image sequences. This is achieved by sampling the texture of the model into the UV domain which remains constant over time. This allows video encoding to be employed to further reduce the storage requirement. Figure 7(b) shows the results of encoding the layers into video sequences. Compression using video codecs achieves a greater reduction as it is able to remove spatial and temporal redundancies from the layer sequences. We again observe the same relationship with an increase in storage as the number of layers is increased. The increase slows as the layers are increased as the texture information stored in these layers becomes sparse due to occlusions. For $L = 3$ with video resolution of 512 and 1024, the layered texture representation reduces the storage requirement by 95% and 90%, respectively.

4.3 GPU Texture Memory

The GPU memory requirement is assessed by calculating the space allocated on the GPU for the textures and other resources required by the rendering pipeline. As a bare minimum, the FVVR requires the M captured frames and M depth buffers, whereas the LFVVR



(a) Image-based storage reduction (b) Video-based storage reduction (c) GPU memory reduction

Figure 7: The storage and memory footprint when varying the number of layers (L) for different layer sizes. (a) Reduction in storage for layers stored as images. (b) Reduction in storage for layers stored as video. (c) reduction in memory when compared to the FVVR

requires L layers, but no off-screen buffers. Figure 7(c) shows the percentage reduction of the LFVVR with respect to the FVVR when using different numbers of layers, L, and for different layer sizes. It can be seen that as L is increased, there is less of a reduction in between the LFVVR and FVVR for all layer sizes. The FVVR requires a fixed amount of memory whereas the LFVVR requires increasing memory depending on how many layers are used. The difference between the layer is cause by using power of two texture sizes resulting in an four-fold increase between the size steps.

4.4 Rendering Performance

In order to evaluate the performance, we calculate the average frame rate for both methods. Tests were performed using a quad-core computer running at 2.40 GHz with 8GB RAM and an nvidia quadro fx 1700 graphics card with 512MB memory. Using $L = 3$ at a size of 512, the LFVVR technique reported an average frame rate of 20 FPS, compared to an average of 11 FPS for the FVVR. This indicates a 50% reduction in rendering cost for the LFVVR representation verse the FVVR. This performance gain is a result of moving the occlusion testing and projecting texturing offline, meaning the LFVVR can simply look-up the texture from the layers and calculate its contribution.

4.5 Application

An implementation of LFVVR was developed to run in a web browser using WebGL and HTML5 based upon [2]. The 3D data and layered textures are stored server-side. These are then downloaded by the client computer where free-viewpoint rendering can be performed locally. An example is shown in Figure 8. This application demonstrates that the layered representation reduces the data transfer and computational cost to enable free-viewpoint rendering on a web-platform.

5. CONCLUSION

This paper introduced a layered texture map representation for efficient storage and view-dependent video-based rendering from multiple view video capture. The layered representation re-samples the images in a hierarchy of texture maps based on the model surface visibility, with the first layer representing the best camera view. The representation allows pre-computation of occlusions and projective texturing for efficient storage and rendering. Evaluation on multi-view sequences of actor performance shows that the storage foot-print is reduced by an order of magnitude, 90%, without loss of visual quality compared to the original foreground images. The layered representation allows pre-computation of visibility, occlusion and projective texturing reducing the rendering cost by 50%. Layered representation is demonstrated to reduce the data transfer and rendering costs to allow free-viewpoint rendering at interactive rates on a WebGL in an HTML 5 browser with textures transferred over a network.

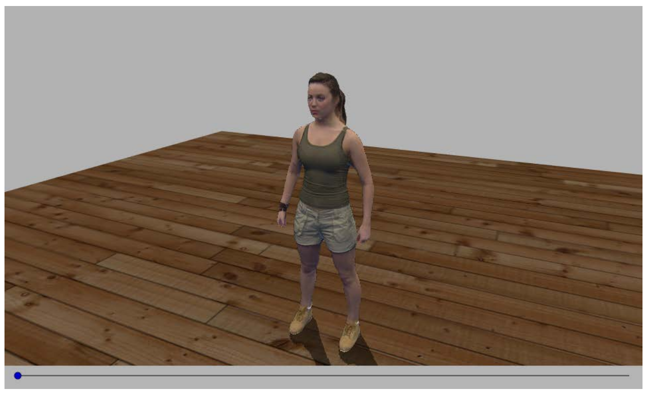


Figure 8: WebGL implementation

Acknowledgements

This work was supported by the EPSRC and the BBC through an iCase studentship and the EU FP7 project RE@CT.

6. REFERENCES

[1] C. Budd, P. Huang, M. Klaudiny, and A. Hilton. Global non-rigid alignment of surface sequences. *International Journal of Computer Vision*, pages 1–15, 2012.

[2] C. Budd, P. Schuebel, and O. Grau. Web delivery of free-viewpoint video of sport events. *NEM2012*.

[3] S. E. Chen and L. Williams. View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 279–288, New York, NY, USA, 1993. ACM.

[4] P. Debevec, Y. Yu, and G. Boshokov. Efficient view-dependent image-based rendering with projective texture-mapping. Technical report, Berkeley, CA, USA,

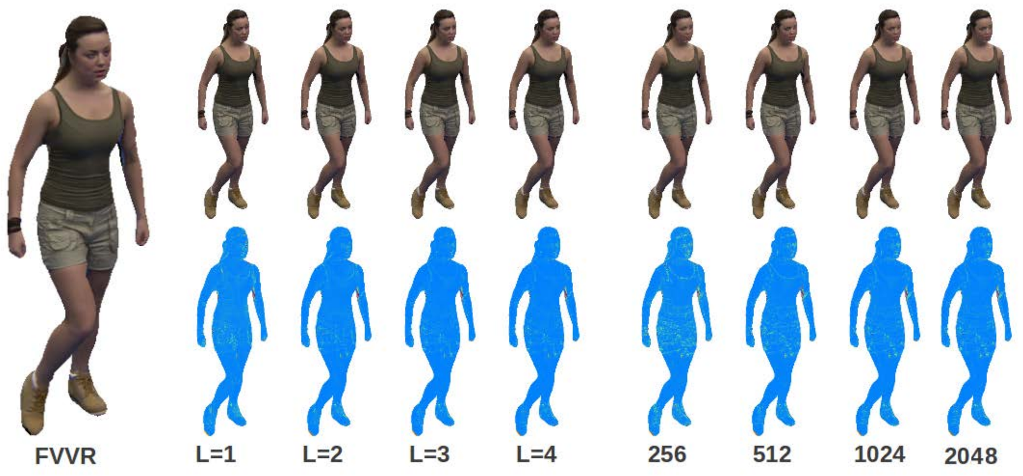


Figure 9: Rendering results using a variety of different rendering settings and texture map sizes.

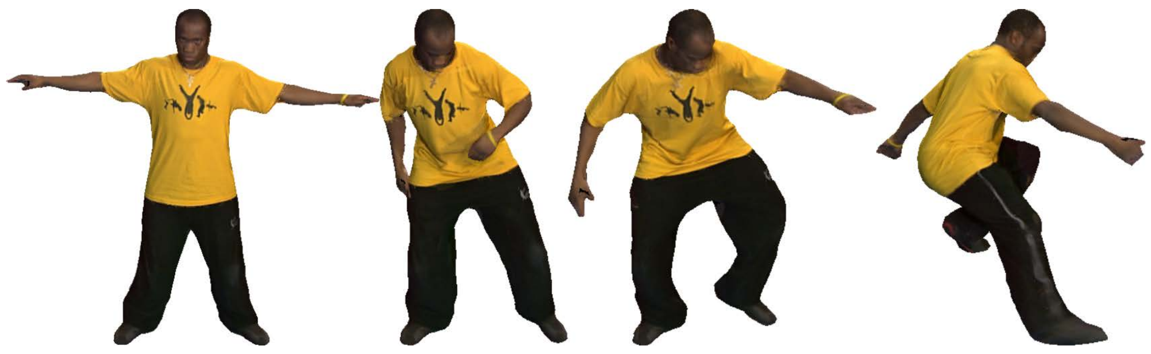


Figure 10: JP character rendered using layered texture maps.

1998.

[5] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 11–20, New York, NY, USA, 1996. ACM.

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

[7] B. Goldluecke and D. Cremers. Superresolution Texture Maps for Multiview Reconstruction. *Science*.

[8] J.-y. Guillemaut and A. Hilton. Joint Multi-Layer Segmentation and Reconstruction for Free-Viewpoint Video Applications. *Signal Processing*, pages 1–27, 2008.

[9] T. Kanade. Virtualized reality: constructing virtual worlds from real scenes. *IEEE Multimedia*, ra-3(4):323–47, 1997.

[10] V. Lempitsky and D. Ivanov. Seamless mosaicing of image-based texture maps. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1 –6, june 2007.

[11] J. Starck and A. Hilton. Model-based multiple view reconstruction of people. *Proceedings Ninth IEEE International Conference on Computer Vision*, 2:915–922, 2003.

[12] J. Starck and A. Hilton. Surface Capture for Performance-Based Animation. *IEEE Computer Graphics and Applications*, pages 21–31, 2007.

[13] J. Starck, J. Kilner, and A. Hilton. Free-viewpoint video renderer. *Journal of Graphic and Techniques*, pages 1–15, 2008.

[14] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

[15] L. Xu, E. Li, J. Li, Y. Chen, and Y. Zhang. A general texture mapping framework for image-based 3d modeling. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 2713 –2716, sept. 2010.

[16] G. Ziegler, H. Lensch, N. Ahmed, M. Magnor, and H.-p. Seidel. Multi-video compression in texture space. In *International Conference on Image Processing*, pages 2467–2470, 2004.