# Online Interactive 4D Character Animation

Marco Volino[*] Peng Huang [†] Adrian Hilton [‡]
Centre for Vision, Speech & Signal Processing,
University of Surrey, UK

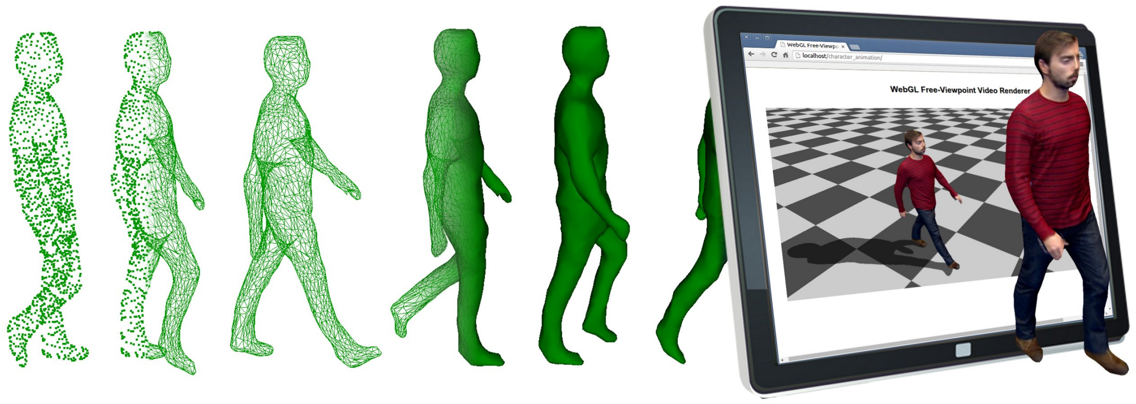

localhost/character_animation/
WebGL Free-Viewpoint Video Renderer

**Figure 1:** *4D interactive character from reconstructed geometry points to textured mesh delivered via a WebGL-enabled browser.*

## Abstract

This paper presents a framework for creating realistic virtual characters that can be delivered via the Internet and interactively controlled in a WebGL enabled web-browser. Four-dimensional performance capture is used to capture realistic human motion and appearance. The captured data is processed into efficient and compact representations for geometry and texture. Motions are analysed against a high-level, user-defined motion graph and suitable inter- and intra-motion transitions are identified. This processed data is stored on a webserver and downloaded by a client application when required. A Javascript-based character animation engine is used to manage the state of the character which responds to user input and sends required frames to a WebGL-based renderer for display. Through the efficient geometry, texture and motion graph representations, a game character capable of performing a range of motions can be represented in 40-50 MB of data. This highlights the potential use of four-dimensional performance capture for creating web-based content. Datasets are made available for further research and an online demo is provided [1].

**CR Categories:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.3.8 [Computer Graphics]: Three-Dimensional Graphics and Realism—Applications;

**Keywords:** WebGL, Performance Capture, Motion Graphs, Character Animation, 4D Video

[*]e-mail:m.volino@surrey.ac.uk
[†]e-mail:peng.huang@surrey.ac.uk
[‡]e-mail:a.hilton@surrey.ac.uk
[1]Website: http://cvssp.org/projects/4d/web3D/

## 1 Introduction

Four-dimensional performance capture (4DPC) enables the acquisition of digital assets that can be used in broadcast, film and game production. This technology has many advantages over traditional animation pipelines as it allows the simultaneous capture of natural motion and appearance without hundreds of man hours currently required by skilled artists. State-of-the-art computer vision algorithms make it possible to acquire three-dimensional (3D) geometry with millimetre scale accuracy [Furukawa and Ponce 2010]. Combining these algorithms with recent advances in geometry processing allow a temporally consistent mesh representation to be extracted from a set of independently reconstructed meshes. This results in a compact representation for time-varying geometry [Budd et al. 2012b]. It has also been demonstrated that this *temporally consistent* representation is efficient for character animation purposes [Casas et al. 2013].

In this paper, a framework is presented which allows the creation of realistic characters that can be interactively controlled in a web browser (Figure 1). Data is captured and processed offline resulting in compact representations of geometry and dynamic appearance. This data is stored on a server, downloaded by the client application and rendered locally using WebGL. A parametric motion graph-based character animation engine is used to provide high-level control of the character demonstrating that *4D Video* data can be used in the production of web-based games. In this work, 4D Video is defined as a temporally consistent geometry representation combined with the camera images capturing the dynamic appearance and geometry of the human performance.

WebGL is a cross-platform, Javascript-based API for rendering 3D graphics natively in a web-browser (e.g. Google Chrome, Mozilla Firefox). It is also becoming increasingly supported by mobile devices (e.g. smart phones and tablets). These properties make WebGL an ideal platform for the delivery of interactive 4D video content and open this technology up to the widest possible audience.

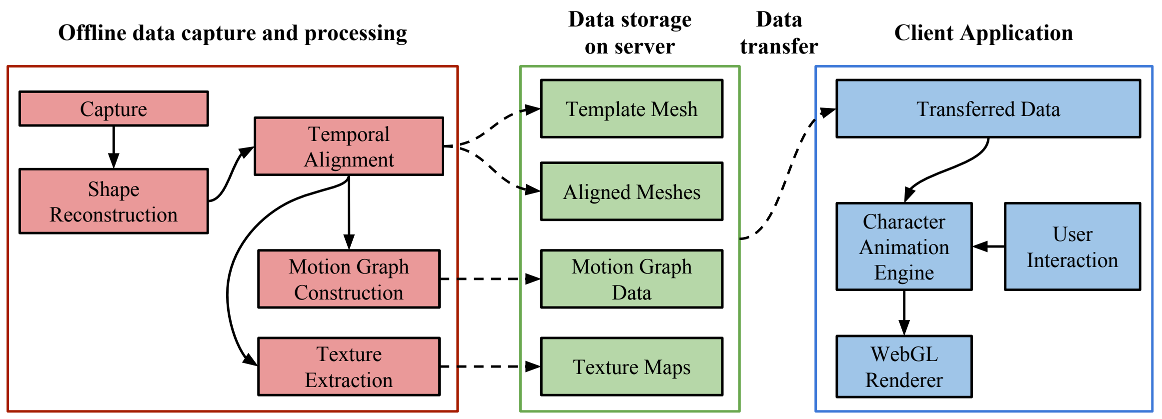


**Figure 2:** *The system consists of three main stages: offline data processing (section 3.1) resulting in geometry, texture map and motion graph data. This data is stored on a server and transfered to a client application (section 3.2).*

## 2 Background

In the last decade, Surface Performance Capture (SurfCap) has been developed to capture geometry, dynamic appearance and motion of the human body from a multi-camera setup [Starck and Hilton 2007]. Different from Motion Capture (MoCap) techniques which capture skeletal motion and re-target to virtual characters conveying a sense of realistic motion for authoring animation in games and film. Performance Capture [de Aguiar et al. 2008; Vlasic et al. 2008; Starck and Hilton 2007; Theobalt et al. 2007] aims to reproduce the fine details of human performance (e.g. wrinkles in clothing and the motion of hair) and reproduce both realistic motion and appearance. The result of Performance Capture is a sequence of 3D meshes and captured camera images. These 3D meshes may not have temporally consistent vertices and topology, making them difficult to store, transfer and re-animate. Recent works on 3D mesh sequence alignment/tracking [Budd et al. 2012b; Allain et al. 2014] enable production of temporally consistent 4D sequences, i.e. a single template mesh deforming over time. Texture generation/extraction from original captured multi-view videos [Volino et al. 2014] allows view-dependent or optimal-views blended texture per frame. 4D videos enable efficient data storage and transmission over the Internet which is also the basis for reanimation of the mesh sequences.

Motion Graph techniques [Kovar et al. 2002; Lee et al. 2002; Arikan and Forsyth 2002] are widely used to synthesise new skeletal movement from a database of skeletal MoCap examples. A graph structure is used to represent possible transitions between different motions while the traversal of the graph results in a concatenative animation. The production of character animation from 4D videos is an analog to conventional MoCap based character animation. Surface Motion Graphs (SMG) [Huang and Hilton 2009] first introduced a framework for concatenative human motion synthesis using unaligned 3D mesh sequences. Transitions are identified between 3D mesh sequences as most similar frames in terms of both 3D shape and motion similarity. Recent work extends to Hybrid Skeletal-Surface Motion Graphs [Huang et al. 2015] considering both 3D shape and appearance similarity and using 4D video as input. Seamless transitions can be created by linear blending 3D meshes of overlapped frames at identified transitions. This work also provide a means to drive a 4D video-based animation with readily available skeletal MoCap data which extends the range of motion that can be produced. 4D Parametric Motion Graphs [Casas et al. 2013] also use 4D video as input. This enables parameterization of motions. A *parametric motion* is defined as a pair of motions with a semantic motion parameter to control the synthesis of new intermediate motion. For instance, with a pair of walk and jog and a speed parameter, a user can create a new motion with any speed between walk and jog. Parametric Motion Graphs also allow responsive transition between parametric motions. Transitions are identified on the fly to balance the responsive time and smoothness. 4D video textures [Casas et al. 2014] provide a run-time optical flow based texture warping to seamlessly blend textures at transitions. However, the computational cost is high for run-time transition identification and optical flow-based texture warping. Due to the WebGL implementation limits, this paper uses fixed transitions between motions for animation control.

Previously, WebGL has been used to deliver free-viewpoint video of sport events [Budd et al. 2012a]. Multiple camera sequences were captured and geometry was reconstructed offline. The time-varying geometry was transferred via HTTP along with the camera images of the captured frames. WebGL was then used to projectively texture the geometry based upon the user selected viewpoint. This framework was limited to the replay of reconstructed data. In contrast, this work applies further offline processing steps to create temporally consistent geometry, texture maps and a motion graph with fixed transition points. This pre-processing also removes the need for online depth testing which requires additional texture buffers and off-screen rendering. The representation allows implementation of an interactive WebGL character animation engine using 4D video.

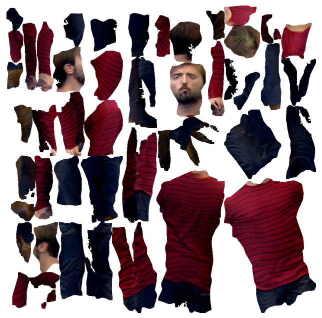
## 3 Character Animation Pipeline

The system consists of three main stages; offline data capture and processing; data storage on a server and transfer; and the client application. Each stage is described below and an overview of the system is shown in Figure 2.

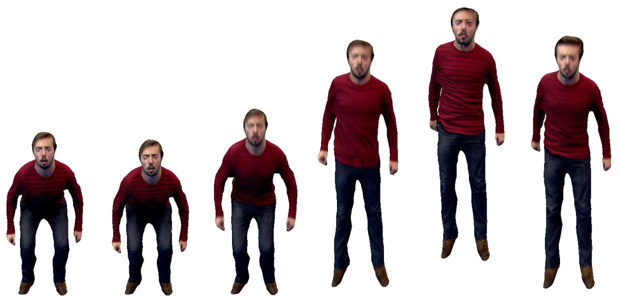
### 3.1 Offline Data Capture and Processing

The capture process is conducted in a dedicated studio using multiple synchronised cameras, see Figure 3a for capture frames. Silhouettes are extracted via chroma keying and are used to reconstruct a visual hull for every frame [Laurentini 1994]. Each visual hull is then refined by matching stereo features across camera images to make the geometry photo-consistent. Stereo refinement adds geometric detail to concave regions of the surface which cannot be recovered using visual hull reconstruction [Starck and Hilton 2007].

**(a)** *Multi-Camera Frame*



**(b)** *Texture Map*



**(c)** *Textured sequence from an arbitrary viewpoint*

**Figure 3:** *(a) A multi-view frame from character Dan walk sequence, frame 5. Note this frame has been cropped based on the extracted silhouette. (b) An extracted texture map for Dan walk sequence, frame 14. (c) Example of texture sequence Dan high jump.*

This results in a set of independently reconstructed meshes which have no correspondence to one another. A temporally consistent mesh representation can be extracted from this collection of unstructured meshes using a non-rigid, global alignment framework based on pairwise matching and Laplician deformation [Budd et al. 2012b].

### 3.1.1 4D Motion Graph Construction

A 4D Motion Graph with 4D video as input is constructed in order to represent possible inter- and intra-sequence transitions. Analogous to motion graph [Kovar et al. 2002] for skeletal MoCap data, allows captured motion sequences being seamlessly concatenated to produce new motion.

To identify potential transitions, we first measure frame-to-frame similarity across all sequences. Both geometry and appearance similarity are considered. A *6D Shape-Colour Histogram* is extracted for each mesh which partitions the space into disjoint cells and counts the number of occupied volume elements falling into each bin together with their RGB colour distribution to construct a 6D histogram as a signature [Huang et al. 2015].

A similarity measure $c(M_r, M_s)$ is defined between two meshes $M_r$ and $M_s$ by minimizing the difference between their corresponding bins with respect to rotation about the vertical axis,

$$c(M_r, M_s) = \min_{\phi} \|H(M_r, 0) - H(M_s, \phi)\|. \quad (1)$$

where $H(M_r, 0)$ and $H(M_s, \phi)$ denote extracted 6D shape-colour histogram for $M_r$ and $M_s$ respectively. The Earth Mover's Distance (EMD) [Rubner et al. 1998] is used to compute the distance between the sparse 6D shape-colour histograms and Equation 1 is minimized in a computationally efficient way: a fine histogram is generated initially at an order of magnitude higher resolution than the desired vertical bin size; the fine histogram is then shifted by the fine bin size and re-binned to a coarse histogram for comparison.

Given a pair of motion sequences (transfer from and transfer to), a transition is determined by a tuple $(m, n, L)$, $m$ and $n$ for identified location in motion sequences and $L$ for overlap length. The optimal can be found as minimising:

$$(m^{opt}, n^{opt}, L^{opt}) = \arg\min_{m,n,L} \sum_{k=-L}^{L} \alpha'(k) \cdot c_{m+k,n+k}. \quad (2)$$

where $\alpha'(k) = min(1 - \frac{k+L}{2L}, \frac{k+L}{2L})$ denotes the weighting for linear blending at transitions. This optimisation is performed as an adaptive temporal filtering with window size $2L+1$ and weighting $\alpha'(k)$ on the precomputed similarity matrix $C$.

Parametric motion is supported within 4D Motion Graphs [Casas et al. 2013]. A parametric motion node contains two or more motion sequences (e.g. a walk and a jog) parameterised to allow generation of any motion in between (e.g. the speed between a walk and a jog). Due to computational cost, the transition between parametric motions is precomputed rather than on the fly as in previous work [Casas et al. 2014]. Fixed transitions are precomputed using the same method as previously described and when motion transferring is required, the motion parameter will first adjust to reach those transitions and then transfer via them. Although multiple transitions are possible, due to computational and data transferring cost for a WebGL application, only the best transition between each pair of motion sequences is kept. The Surface Motion Graph is stored as an XML file for future use. An example of a Surface Motion Graph is shown in Figure 4.

### 3.1.2 Texture Extraction

The texture map at frame $t$ is generated using the following process and can be implemented in OpenGL Shading Language.

**Input:** Template Mesh $M_T$, Mesh $M_t$ at frame $t$, Camera Projection Matrix $\pi_t^i$, Normalised camera direction vector $c_i^d$, Camera Images $\{I_t^i\}_{i=0}^{n_C}$, Rendered Depth Maps $\{D_t^i\}_{i=0}^{n_C}$,

**Process:** Each output pixel is associated with a real or interpolated vertex $v$ consisting of a 3D position $v_p$, normal $v_n$ and texture coordinate $v_u$. The following process is performed on each output pixel of the texture map:

1. Perform a depth test in all capture cameras by projecting $v_p$ into $\{D_t^i\}_{i=0}^{n_C}$ using $\pi_t^i$.
2. Sort all visible cameras based on the angle between $c_i^d$ and $v_n$ in ascending order, e.g. cameras with a more direct view are preferred.
3. For the best N visible cameras, sample colour by projecting $v_p$ into $\{I_t^i\}_{i=0}^{n_C}$ using $\pi_t^i$ and use a weighted average of all N colours based on the previously calculated angles.

**Output:** Texture map $T_t$ for frame $t$

A template mesh $M_T$ is selected and texture coordinates $U$ are created, which define a mapping from the 3D surface to the 2D texture domain. Current solutions to generate $U$ for an arbitrary shape

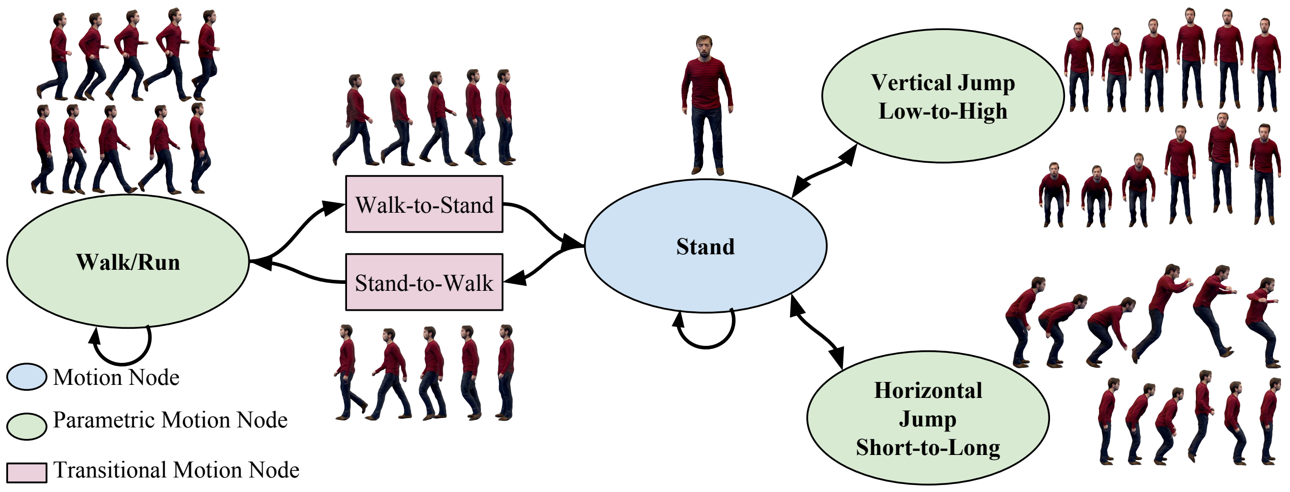


**Figure 4:** *Motion graph example showing how different motion nodes are connected.*

are performed either automatically by positioning cuts on the surface based on curvature [Lévy et al. 2002] or require cuts to be manually defined as in model pelting [Piponi and Borshukov 2000] prior to the surface being flattened. In this work, $U$ is generated by manually defining seams to split the mesh up into semantically meaningful parts (e.g. arms, legs, face, etc) using Blender (version 2.62). This operation is performed once for a character database as all meshes share the same vertex count and mesh topology.

Texture extraction is performed on every frame and maintains the temporal, dynamic appearance of the surface such as wrinkles in clothing and facial expression which are not modeled geometrically. Figure 3b shows an example of an extracted texture map. Figure 3c shows examples of textured model sequences.

## 3.2 Client Application

The application is implemented in Javascript and consists of two main components; a character animation engine (CAE) and a WebGL-based renderer. The client application starts by downloading all the required mesh, texture and motion graph data. The CAE maintains the current state of the character based on the user input. The frame, or pair of frames in the case of a parametric motion, which are required to be displayed are sent to the WebGL renderer.

### 3.2.1 Character Animation Engine

The CAE is implemented based on the previously constructed Parametric Motion Graph and allows user interactive control over motion transfer and motion parameter adjustment. Figure 5 illustrates the flow chart of the CAE: Motion Graph and Motion Database are first loaded from xml files. User interactive control will result in a queue of motion requests. The CAE calls a traverse function to walk through the Motion Graph and each step will point to data in the Motion Database. Depending on whether the frame needs to be blended, single or multiple pairs of mesh and texture data are passed to the WebGL renderer. Playing will finish when no more motion requests are in the queue. But if the current motion is a loop motion, it will continue playing (e.g. a walk cycle). While the animation is playing, the user can adjust motion parameters for the current parametric motion (e.g. when the virtual character is performing a “walk/jog” motion), the user can speed up or slow down and this will immediately change the motion.

### 3.2.2 WebGL Renderer

The resources of the client device are not known prior to rendering. To ensure that the application can run on devices with different capabilities, only the minimum amount of graphics resources are allocated (e.g. data buffers and textures) . Mesh and texture data is stored in client memory and the WebGL buffers are updated as required. The temporally consistent geometry representation allow for only the vertex position and texture map buffers to be updated on a frame-to-frame basis, as the mesh connectivity and texture coordinates remain constant over all frames. Basic shadowing is achieved by positioning a virtual light source in the scene and rendering a depth map from the light source viewpoint. WebGL currently does not support access to the depth buffer so depth values are packed/unpacked into a 24bit representation. When rendering the floor plane each vertex is projected into the depth buffer. If a vertex falls into a region occluded by the characters depth map a shadow effects is applied. This is performed over a 3x3 window to enable soft edges to shadows.

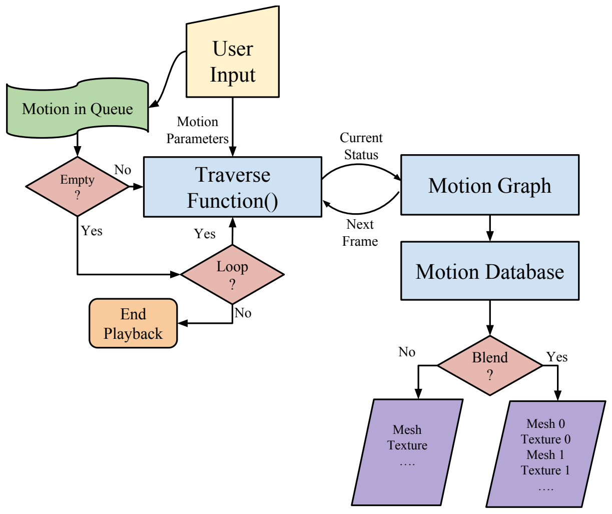


**Figure 5:** *Flow chart of character animation engine showing how motion parameters are used to query the motion graph and motion database based on user input.*

| Dataset | Captured Data (MB) | Frames (Motions) | Processed Data (MB) | | | Polygon Count | Runtime | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Geometry | Texture | Total | | RAM(MB) | Load Time(s)$^+$ | FPS Range |
| Dan | 6800 | 254 (8) | 12 | 27 | 39 | 5330 | 220 | 30-60 | 30-70 |
| Roxanne | 9500 | 432 (10) | 20 | 31 | 51 | 4950 | 350 | 30-60 | 30-70 |
| Ballet | 60300 | 3301 (9) | 304 | 1$^*$ | 305 | 9996 | 1700 | >120 | 30-70 |

**Table 1:** *Overview of datasets and storage requirements of captured and reconstructed data. $^*$Due to the number of frames a single texture map was applied to all frames. $^+$Loading times are dependent of internet connection speed.*

## 4 Results

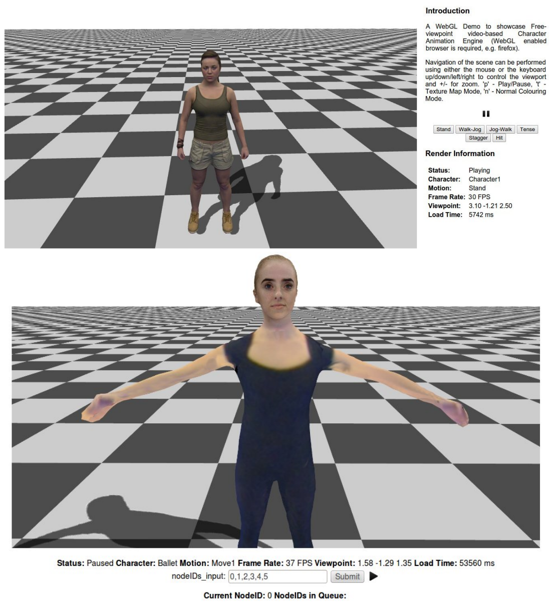
In order to test the framework, three characters were processed and rendered using the WebGL animation engine. An overview of each character scenario is given below.

**Character Dan**
A male character in a red and black sweater, dark jeans and brown shoes was captured performing typical game character motions, e.g. idle, walk, run, horizontal and vertical jumps [Casas et al. 2014]. In this demonstration, the user can interactively control the viewpoint, the state of the character using HTML buttons, and motion parameters when in a parametric motion node (e.g the speed of walk/run, the height of vertical jumps and the length of horizontal jumps). This character requires 30 MB of mesh and texture data. It is made up of eight motions with a total of 254 frames. Once loaded requires 220 MB of RAM. Examples of this character are shown in Figure 7.

**Character Roxanne**
A female character in a green top, camouflage shorts and brown boots was captured preforming typical game character motions including walk, run, tense, hit and stagger. This data is part of the SurfCap dataset [Starck and Hilton 2007]. In this demonstration, the user can interactively control the viewpoint, the state of the character, and motion parameters. The complete character consists of 10 motions, requires 50 MB of data to be transfered, and once loaded requires 370 MB of RAM. Examples of this character are shown in Figure 8.

**Ballet Character**
A female ballet dancer dressed in a black leotard was captured performing nine short (5-10 second) dance segments, starting and ending in a neutral pose. In this example, high resolution geometry of the character's face was captured and reconstructed separately [Blumenthal-Barby and Eisert 2014]. This high resolution face model was then fused to a lower resolution body template and temporal alignment performed [Allain et al. 2014]. In this demonstration, the user can reorder the small dance segments to create a unique dance. Example frames are shown in Figure 9. This is the largest demonstration consisting of 3300 frames, requiring 305 MB of data to be transfered and using 1.7 GB of RAM. To make this demonstration practical, a single texture map was selected and applied to all frames.

All test were conducted on a Dell Optiplex 9010 desktop computer with an nVidia GT 640 graphics card running Ubuntu 12.04 on Mozilla Firefox (version 35.0) and Google Chrome (version 40.0). The rendering frame rate ranged from 30 to 70 frames per second (FPS). It was observed that Mozilla Firefox consistently achieved a higher FPS than Google Chrome in all demonstrations. Loading time is dependent on the Internet connection speed with the Dan and Roxanne characters requiring between 30 seconds to 1 minute. The Ballet example however took several minutes to load and in its current form would probably be unsuitable for general use. Recommendations to address problems with the data size are made in section 5. Examples of the user interface are shown in Figure 6.


Introduction

A WebGL Demo to showcase Free-viewpoint video-based Character Animation Engine (WebGL enabled browser is required, e.g. firefox).

Navigation of the scene can be performed using either the mouse or the keyboard up/down/left/right to control the viewport and +/- for zoom. 'p' - Play/Pause, 't' - Texture Map Mode, 'n' - Normal Colouring Mode.

Render Information

Status: Playing
Character: Character1
Motion: Stand
Frame Rate: 30 FPS
Viewpoint: 3.10 -1.21 2.50
Load Time: 5742 ms

**Status:** Paused **Character:** Ballet **Motion:** Move1 **Frame Rate:** 37 FPS **Viewpoint:** 1.58 -1.29 1.35 **Load Time:** 53560 ms
nodeIDs_input: 0,1,2,3,4,5

**Current NodeID:** 0 **NodeIDs in Queue:**

**Figure 6:** *Example of user interface for Roxanne character (top) and Ballet character (bottom)*

## 5 Conclusions

This paper has presented a framework for creating 4D characters and allowing interactive animation to be performed in a WebGL enabled web browser. This was achieved through the development of a Javascript-based character animation engine and a WebGL rendering engine. A key challenge of this work is efficient representation of geometric data which is achieved using a temporally consistent geometry representation and by combining the multiple-camera images into a texture map. Three demonstrations based on this framework were presented with each character expressing a different set of motions. The motion of a typical game character can be represented in a few hundred frames which equates to between 40-50MB of data.

Future work will investigate further reductions in data size with an emphasis on maintaining visual quality and how to efficiently transfer data. This could include; Geometry compression algorithms; Improvement of texture maps through a super-resolution approach [Goldluecke and Cremers 2009; Tsiminaki et al. 2014]; Compression of dynamic texture maps to maintain realism.

## Acknowledgments

This research was supported by the EU FP7 Project RE@CT (grant agreement n$^o$ 288369) and EPSRC/BBC industrial case studentship award '3D Video Editing'. The temporally consistent geometry of the Ballet character, featuring a high resolution face fused with a lower resolution body, was provided by RE@CT project partners.

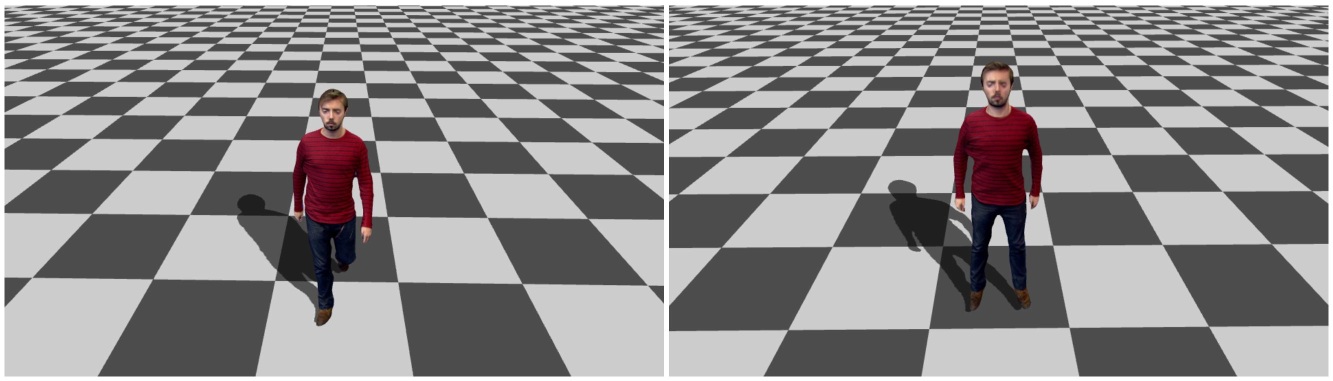


**Figure 7:** *Dan Character Example. The current motion of the character is controlled using HTML buttons below the viewer. Parametric nodes are controlled using assigned keys and the viewpoint is selected using either the mouse or keyboard controls.*

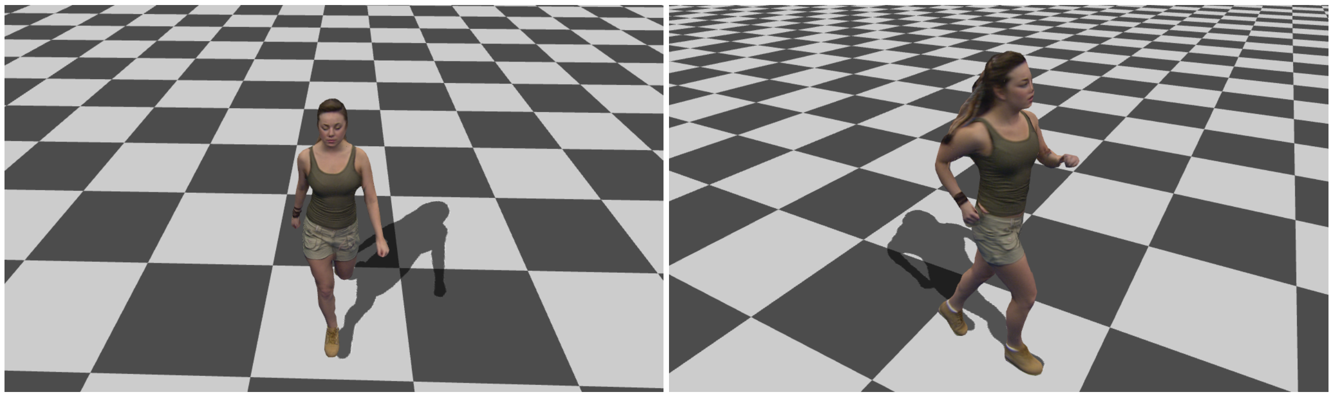


**Figure 8:** *Roxanne Character Example: The character and viewpoint can be interactively controlled by the user. Parametric nodes (e.g. speed of walk/run) are controlled using the keyboard. This example consists of six motions including stand, walk, run, stagger, tense and hit.*

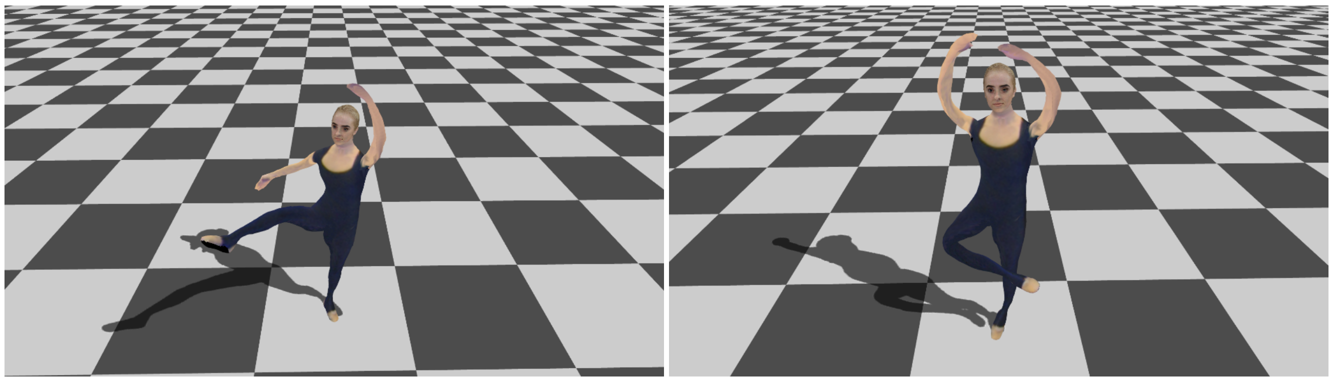


**Figure 9:** *Ballet Example: Users can create a unique dance sequence by concatenating dance moves together.*

## References

ALLAIN, B., FRANCO, J.-S., BOYER, E., AND TUNG, T. 2014. On Mean Pose and Variability of 3D Deformable Models. In *ECCV 2014 - European Conference on Computer Vision*, Springer, Zurich, Switzerland.

ARIKAN, O., AND FORSYTH, D. 2002. Interactive motion generation from examples. *ACM Transactions on Graphics 21*, 3 (July), 483–490.

BLUMENTHAL-BARBY, D. C., AND EISERT, P. 2014. High-resolution depth for binocular image-based modeling. *Computers and Graphics 39*.

BUDD, C., GRAU, O., AND SCHÜBEL, P. 2012. Web delivery of free-viewpoint video of sport events.

BUDD, C., HUANG, P., KLAUDINY, M., AND HILTON, A. 2012. Global Non-rigid Alignment of Surface Sequences. *International Journal of Computer Vision 102*, 1-3 (Aug.), 256–270.

CASAS, D., TEJERA, M., GUILLEMAUT, J., AND HILTON, A. 2013. Interactive animation of 4d performance capture. *Visualization and Computer Graphics, IEEE Transactions on 19*, 5, 762–773.

CASAS, D., VOLINO, M., COLLOMOSSE, J., AND HILTON, A. 2014. 4d video textures for interactive character appearance. *Computer Graphics Forum (Proc. Eurographics 2014) 33*, 2.

DE AGUIAR, E., STOLL, C., AND THEOBALT, C. 2008. Performance capture from sparse multi-view video. *ACM Transactions on ...*, 1–10.

FURUKAWA, Y., AND PONCE, J. 2010. Accurate, dense, and robust multiview stereopsis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 32*, 8 (Aug), 1362–1376.

GOLDLUECKE, B., AND CREMERS, D. 2009. Superresolution texture maps for multiview reconstruction. *IEEE 12th International Conference on Computer Vision* (Sept.), 1677–1684.

HUANG, P., AND HILTON, A. 2009. Surface motion graphs for character animation from 3d video. In *SIGGRAPH 2009: Talks*, ACM, New York, NY, USA, SIGGRAPH '09, 56:1–56:1.

HUANG, P., TEJERA, M., COLLOMOSSE, J., AND HILTON, A. 2015. Hybrid skeletal-surface motion graphs for character animation from 4d performance capture. *ACM Trans. Graph. 34*, 2 (Mar.), 17:1–17:14.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '02, 473–482.

LAURENTINI, A. 1994. The visual hull concept for silhouette-based image understanding. *Pattern Analysis and Machine Intelligence, IEEE ...*.

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics 21*, 3, 491–500.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics ( ...*, 362–371.

PIPONI, D., AND BORSHUKOV, G. 2000. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, ACM Press, New York, New York, USA, 471–478.

RUBNER, Y., TOMASI, C., AND GUIBAS, L. J. 1998. A Metric for Distributions with Applications to Image Databases. *Computer Vision, 1998. Sixth International Conference on*, 59–66.

STARCK, J., AND HILTON, A. 2007. Surface capture for performance-based animation. *Computer Graphics and Applications, IEEE 27*, 3 (May), 21–31.

THEOBALT, C., AHMED, N., LENSCH, H., MAGNOR, M., AND SEIDEL, H.-P. 2007. Seeing people in different light–joint shape, motion, and reflectance capture. *IEEE transactions on visualization and computer graphics 13*, 4, 663–674.

TSIMINAKI, V., FRANCO, J., AND BOYER, E. 2014. High Resolution 3D Shape Texture from Multiple Videos. *International Conference on ...*.

VLASIC, D., BARAN, I., MATUSIK, W., AND POPOVIĆ, J. 2008. Articulated mesh animation from multi-view silhouettes. *ACM Transactions on Graphics 27*, 3 (Aug.), 1.

VOLINO, M., CASAS, D., COLLOMOSSE, J., AND HILTON, A. 2014. Optimal representation of multiple view video. In *British Machine Vision Conference*.