EC FP7 project n° 288369

IMMERSIVE PRODUCTION AND DELIVERY OF INTERACTIVE 3D CONTENT

DELIVERABLE D5.1.2
UPDATE OF SPECIFICATION OF PLATFORM, ANIMATION MODULES AND INTERFACES

Contractual Date of Delivery:	July 31, 2013
Actual Date of Delivery:	July 31, 2013
Work Package:	WP5
Dissemination Level:	Public
Nature of Deliverable:	Report
Authors:	Graham Thomas, Jim Easterbrook (BBC), Peter Eisert, Markus Kettern, David C. Blumenthal-Barby (Fraunhofer), Edmond Boyer (INRIA), Dan Casas, John Collomosse, Adrian Hilton, Peng Huang (UoS), David Knossow (Artefacto), Emilio Maggio, Paul Smyth (OMG)
Editor:	David C. Blumenthal-Barby (Fraunhofer), Hélène Waters (BBC)

Classification and Approval

Classification: Public

This document has the status 'Public' and may be published as a whole including this page. Publication of parts of the document must be in unanimous agreement within the RE@CT Steering Board and subsequent EC approval/agreement.

Disclaimer

Acknowledgements

All partners of the RE@CT project contributed to this report.

The work described in this report was partially funded by the European Commission under the 7th Framework Programme for Research (2007-2013)

Abstract

This document is the updated specification of the platform, animation modules and interfaces of RE@CT. It is based on deliverable D5.1.1, which described the initial specification at an early stage of the project. Designed as a rolling document, changes were made to the initial specification where necessary, and the original text was kept otherwise. Deliverable D5.1.2 reflects technology decisions made after the first two test productions, and adds algorithms and implementation details to issues left open in D5.1.1. The initial specification proved to be practicable and robust, and thus remains valid for many parts of the project.

Table of Contents

List of Illustrations

1. WP2 Studio Capture System

The major aim of the studio system is to provide high-quality capture of human actions that preserves the full repertoire of body language as well as highly detailed facial expressions used in acting. The project will develop a purely camera-based acquisition system for whole body and facial 3D video. In order to allow for simultaneous development of temporally consistent data representations (WP3) and actor animation algorithms (WP4), a hybrid performance capture system will be developed, combining multi-view video and state-of-the-art marker-based motion capture technology. In addition to the stationary camera array, a set of multiple pan/tilt/zoom cameras or alternatively head-mounted cameras will simultaneously capture high-resolution views of the head. In order to help actors interact with virtual objects, a novel feedback system will be investigated that correctly projects virtual entities into the studio environment from the actor's perspective.

1.1. The capture studio

Although all partners have local access to multi-camera capture facilities that will be used to facilitate internal R&D activities, the consortium uses the BBC studio in London as the reference RE@CT capture location. The studio was and will be used to test integration of different video and motion capture technologies and all gathered data will be available to the partners to test algorithms and systems developed in the other work-packages.

The studio has built-in facilities for hardware synchronisation, a professional lighting system and studio truss system to mount video hardware from various partners. To facilitate silhouette extraction the sides and the floor can be covered with retro-reflective clothing. The size of the studio (approximately 6 x 10 metres) is sufficient for the scope of the RE@CT project where we expect to capture one/two characters at the initial stages, and eventually up to four characters, at the end of the project.

1.2. Whole body capture system

As mentioned in the previous section, RE@CT will target the capture of whole body movements of up to two actors. This will be achieved by a system of fixed cameras positioned at the edges of the capture volume. The exact positioning will depend on the type of scenario captured on the day, on the number of actors and on the script. The cameras will be provided by BBC and OMG. Two systems will be developed: a preliminary hybrid system where the capture of action is aided by a commercially available Vicon motion capture system, based on 3D localisation of retro-reflective markers; and a final system where no special markers are used. The specifications for the marker-less system as well as for the hybrid system are outlined below.

1.2.1. Marker-less setup

The BBC multi-camera capture studio in London is currently equipped with 12 (but can be more) HD video cameras (see Illustration 1 (a)) with the following specifications:

1. 10x HD Sony X300 - 1920x1080 50i or 25p (or 60i/30p) (brick cameras)
2. 1x Canon HD Camcorder - 1920x1080 50i (interlaced)
3. 1x Sony DCP-9100p - SD 720x576 50i or 25p

The cameras produce RAW YUV files (a single .yuv file per camera, encoded as 8-bit 4:2:2 YCrCb).

In addition to these cameras OMG will provide 12 high frame rate Bonita video cameras (see Illustration 1 (b)) from their own production with 720p resolution at 120fps. Although the image

quality of those cameras is lower than that from a 3CCD sensor, high frame rate capture can provide better data with fast motions. The RAW output of the Bonita cameras is Bayer encoded colour AVI files.

Synchronisation between the two systems will be achieved by means of industry standard gen-lock signals. To this end, preliminary tests have already taken place to check for compatibility between the two systems. To further facilitate data integration, synchronisation via Time-Code information will be investigated as well.

To calibrate the system in terms of camera location and lens distortion models, prior work from BBC and OMG will be used. The calibration procedure involves waving a wand with multiple LEDs in front of the cameras. Prior knowledge of the LED positioning is used by a bundle adjustment algorithm to estimate the parameters of the lens distortion model, the camera intrinsic parameters (i.e. focal length, image format, and principal point), and the camera position and orientation. Lens distortions will then be compensated. This means all images used in later processing stages will be corrected. This simplifies the data exchange and data processing.

(a)

(b)

(c)

Illustration 1, sample video cameras used by the RE@CT capture studio. Standard HD video cameras (a). High frame rate video cameras (b). Infrared motion capture cameras (c).

1.2.2. Hybrid setup

The specifications of the hybrid motion capture system are the same as the marker-less system described in the previous section but with the addition of an extra set of specialized cameras from OMG to track a set of retro-reflective markers (see Illustration 1 (c)). The optical motion capture system uses reflected infrared light to estimate the 3D position of the markers with sub-millimetre accuracy. The marker positions add an element of reliability to the system and can find application at different levels of the data processing pipeline. These positions can be used to drive an articulated

model of the actors and to compute limb poses and joint angles or as constraints for the 3D mesh generation and motion analysis algorithms.

For the RE@CT test productions we use up to 14 optical cameras with on-board marker tracking. The cameras have a maximum resolution of 4 megapixels and up to 800 frames per second.

Synchronization with the video system is achieved via Genlock, and optionally Time-Code signals, as with the video cameras described above. The same LED wand will be used for camera calibration.

1.3. Face capture system

To convey accurate and entertaining animations, capturing the actors' expressions is of utmost importance. The data to achieve these goals will be captured using two different approaches: a head-mounted capture rig from OMG and a system of automatically controlled PTZ[1] cameras which will be developed by BBC during the project.

1.3.1. Head mounted cameras

For the initial capture activities of RE@CT, OMG will provide two head-mounted camera systems (see Illustration 2 for an example of a current prototype). Each system is composed of four monochrome cameras positioned as two stereo pairs with a 720p image format and a refresh rate of 60 frames per second. The videos are compressed as H264 AVI files and stored in a logger drive also worn by the actor. The device synchronizes with the rest of the capture system (see previous sections) using a jam-sync Time-Code signal and then runs freely using an internal clock as reference.

Software is available that performs a frame-by-frame camera calibration procedure to compensate for relative movements between the two arms and with respect to the face.

Illustration 2, example of head-mounted system used to capture actors' expressions. Visible in front of the actor's face are four cameras arranged as two stereo pairs.

[1] PTZ: pan tilt and zoom

1.3.2. Dedicated head capture with pan/tilt/zoom cameras

The overall aim of head/face processing in RE@CT is to enable viewpoint modification for head/face close-ups and to provide spatio-temporally consistent and semantically annotated video material for re-animating head/face close-ups in the motion graph framework. The head capture process must capture enough data of sufficient quality for these goals.

While the initial capture setup involves a head-mounted camera rig, as described above, the head capture process will evolve towards non-intrusive capture during the project. For non-intrusive capture of the actors' heads and faces, footage is shot on-scene by multiple dedicated cameras, recording at least in full HD (1080p) quality. The cameras have pan/tilt/zoom (PTZ) capability in order to focus on the actors' heads/shoulders and provide high resolution facial footage, which cannot be recorded by the "body" cameras covering the overall scene. They are operated either manually by trained personnel or automatically by robotic camera heads.

The number of head cameras required depends on the number of actors involved in the scene and on the degree of freedom required for re-rendering. Ideally, each head should be captured covering a full 360 degrees with some overlap between neighbouring views. In practice, coverage from the front over an angle of 120 degrees is most relevant, allowing the rendering of half-profile views from both directions. 180 degrees must be covered if full profiles are required. The PTZ cameras must be temporally synchronized with the body-capture cameras and they must be calibrated.

The BBC Research studio is equipped with cameras with electronic feedback on zoom settings. Additionally, an optical calibration system is available for extrinsic camera parameters, consisting of markers on the studio ceiling which are recorded by a second camera mounted on the PTZ camera. The optical tracking system can be combined with any broadcast camera mounted on a studio camera pan-tilt pedestal or tripod. The operation would then be manual and handled by an operator.

In D5.1.1 it was proposed to mount one or more of the PTZ cameras on a robotic pan-tilt head. The head would be controlled automatically with a real-time head-tracker. The head-tracker is based on a 3D reconstruction of the actor using silhouette cut from at least 5 inputs. The control component for the pan/tilt head attempts to derive smooth predicted paths from tracked head positions, with full speed 'goto' where necessary. Maximum acceleration and velocity are programmable; the robotic head itself is capable of a lot more than required for the project. Currently, an acceleration limit of 30 degrees/second and a velocity limit of 100 degrees/second is used, which has not been reached in practise yet.

Tests so far have failed to achieve satisfactory results using automated tracking. The latency of image capture, 3D reconstruction and robotic head drive result in the camera failing to keep track of even quite modest movements. If these problems cannot be rectified then all the PTZ cameras will be manually controlled.

1.3.3. Pre-modelling of the head

To support the computation of 3D information from the PTZ sequences, a detailed, fully textured 3D model is generated from the actors involved in the shoot in advance. The model is captured with Fraunhofer HHI's image-based head-capture rig and computed with algorithms available at HHI. The rig comprises up to five narrow-baseline SLR camera pairs at 50mm focal length (full frame equivalent) as well as studio flash lighting. The 3D model covers a head-to-throat portrait of the person over an angle of 120 degrees (frontally). It has high-resolution textures (18 megapixels) and detailed 3D geometry.

The benefits of head processing from multiple 3D models per actor, each showing a different facial expression, are under investigation by HHI. During the capture sessions at the BBC research studios, four large datasets of three different actors were captured, with up to 50 different facial expressions

shot simultaneously from up to five viewpoints. Capturing multiple models in the 3D capture rig is straightforward as the capture process is instantaneous.

From the captured stereo image pairs, detailed depth meshes are reconstructed with an algorithm based on non-linear warp optimization. To obtain a single closed mesh of the head for each facial expression, the depth maps need to be merged. Several merging strategies are implemented and investigated:

- Merging with a cylinder mesh topology emulates the operation of a classic head scanner: The depth meshes are sampled along the normal directions of a cylinder aligned with the geometry's principal axis.
- Model-based merging uses a learned parametric model of head geometry, a morphable model, as topology source for the merged mesh. This requires, besides the model itself, algorithms for fitting the model to the reconstructed geometry before the depth meshes are sampled. A morphable head model developed at HHI is refined for use in RE@CT, and the required fitting algorithms are implemented.

For both merging strategies, ambiguities in regions where multiple meshes overlap must be resolved. This is achieved with a discrete optimization algorithm based on Message Passing in Factor Graphs, which takes into account the reconstruction quality and the smoothness of the resulting merged mesh.

1.4. 3D full-body mesh generation

The 3D geometry of the captured action is computed frame-by-frame using a robust implementation of a visual hull computation. This requires known camera parameters, as computed by the system calibration and a segmentation of the scene into background and foreground, i.e. the actor's silhouette. We use chroma-keying for the segmentation, supplemented by difference keying in areas that have no chroma-keying background. This procedure results in a preliminary 3D volumetric and surface data that can be used for previewing and will be refined by the offline algorithms developed in WP3.

1.5. Data exchange

To facilitate the exchange of data, the RE@CT partners have agreed to a set of common data formats. While the original RAW data will be available on request the consortium will encourage each partner to share the data according to the following directives.

1.5.1. Sharing videos

The capture systems described in the previous section can generate a formidable amount of data. Each camera produces up to 100MB/sec of video. For this reason the exchange of raw data is neither practical nor desirable.

For short sequences the partners will exchange video data stored as single Portable Network Graphics (PNG) files compressed using lossless encoding. To store timing information the filename will include a frame number initialised at some time of the day (e.g., mid-night) at predefined clock rate (e.g., 135 MHz).

The issue with PNG files for longer sequences is that the number of files becomes problematic for the operating system to handle. Also, the compression achieved by lossless encoders is not sufficient and data exchange remains problematic.

To exchange longer sequences the partners in charge of data capture will compress the video using H264. H264 was chosen as it is a global standard and decoding software is easily available. To preserve details in images low compression profiles are suggested.

Alpha mattes used for silhouette extraction will be stored as PNG file sequences. As alpha mattes are highly redundant, file sizes are not an issue, even for PNG sequences.

1.5.2. Lens distortion

The lens distortion models used by different partners are very similar, with radial distortion modelled by even order polynomials (up to fourth order). However there are differences on whether the direct or inverse transformation is modelled in closed form.

Following discussions at the RE@CT second consortium meeting, it was agreed that the best and simplest option would be for partners to share undistorted video sequences. The advantage of this approach is that it is not necessary to share and provide support for conversion code. However the conversion might reduce the quality of the original video. We have assessed this risk and we expect the loss to be limited and not sufficient to affect the algorithms part of the RE@CT production pipeline.

1.5.3. Camera calibration

As expected, all partners represent camera calibration information in the standard way as a set of intrinsic camera parameters (i.e., image size, principal points and focal length) and extrinsic parameters (i.e. camera position and orientation). Nevertheless there are slight differences in terms of data conventions. For example 3D camera rotation can be represented as a quaternion, an axis-angle vector or a rotation matrix.

To make the representation as free from conventions as possible the partners have agreed to share camera calibration information as a set of 3-vectors (position, viewing direction and up-direction) + focal length, or alternatively a 3x4 projection matrix stored either in a xml document or a text file.

1.5.4. 3D meshes

The output of WP2 will be a set of non-temporally consistent 3D meshes. Several partners have proposed and implemented algorithms to generate 3D meshes from multi-view video. To exchange data the partners have agreed to use the open Wavefront .OBJ file format [2]. This simple format represents geometry only as a concatenated list of vertices, texture coordinates and faces.

1.5.5. Metadata

All meta-information will be stored in human-readable XML files. Directory structures with naming conventions are used to structure datasets. Specific structure and XML data formats are described in sections 2 and 3.

1.6. Immersive production system

The processing modules of WP3 that extract the 3D geometry of actors are based on the assumption that the scene can be segmented and ideally actors are captured in isolation in a controlled studio environment with chroma-keying facility. This raises the issue that actors have no visual reference: it is very difficult even for trained actors to interact with virtual objects they do not see. The unknown

position of another person/object may lead to an incorrect pose or gaze by the actor. Similarly, the temporal synchronisation with movements, events, or gestures is difficult to achieve without any feedback.

In order to simplify the capturing process and to obtain more satisfactory results, task T2.4 addresses the further investigation of a feedback system previously developed to help the production of special effects. It provides the actor with visual feedback of virtual humans/objects from the interactive platform. The virtual scene is rendered with accurate timing and projected into the real studio environment without interfering with the capturing. The 3D position of the actor is considered so that the projected entities appear in the correct direction and help them to perform correctly in the virtual environment. This task addresses the problems of interactive programme-making and will adapt the previously developed visual feedback system to the new use case scenarios.

1.6.1. Retro-reflective cyclorama

A studio with chroma-keying facility usually relies on having a coloured background and floor, so that the actors and props can be keyed onto the virtual background.

In order to generate a good key signal with low noise, it is necessary to have the coloured background illuminated brightly and evenly. This gives rise to several problems, particularly when used in large studios. BBC R&D previously developed a retro-reflective background cloth, in collaboration with a manufacturer to tackle problems such as the lack of homogeneous lighting (see Illustration 3). The cloth is coated with a large number of microscopic half-silvered glass beads, randomly orientated. Light entering the front of a bead is returned along the path it came from. By mounting a ring of lights around the camera lens having the chosen key colour, the camera will see the background as having the desired uniform colour, regardless of the setting of the studio lights. The reflectivity of the cloth is sufficiently high that a low level of illumination from the LEDs mount on the camera is sufficient, and the amount of light scattered from actors is generally negligible.

We have found that the reflective cloth works particularly well as a combined projection screen and chroma-key background. This is because the camera always sees the key-coloured light much more strongly than other light scattered from the cloth, since the key light is mounted close to the camera lens. Conversely, the performer sees a roughly equal proportion of all light landing on the cloth, as the cloth scatters a significant proportion of incident light. Thus it is possible to give the performer a relatively high-contrast image whilst the camera sees an image composed mostly of the key colour. Illustration 3 shows an example of a projection used to visualise virtual objects for special effects production using view-dependent projection techniques as described below.

Illustration 3, a view-dependent projection onto reflective cloth

1.6.2. Physical set-up

To give actors visual feedback, data projectors need to project onto screens placed in the direction the actor is facing, and with which they interact. The screens are either wall surfaces or the keying material (retro-reflective cloth) in this case. Usually the projection surface will include at least the wall the actor is facing. In more complex actions, projection onto two, three or all surrounding walls and the floor might be required. Depending on the shape and size of the studio and the actual requirements regarding the scene complexity, there can be any number of projectors from one to 5 (4 walls + floor) or even more if more than one projector per wall/floor is needed.

In order to work without interference with the LED-based chroma-keying, the data projectors should not be mounted too close to cameras. In practise we found that a minimum separation of approximately 1m is sufficient.

1.6.3. Software

To give the actor an immersive visualisation of the scene, a view-dependent rendering is needed. This requires a rendering system able to render a view of the scene depending on a) the position of the actor's head; b) the position of the projector and c) the screen size and position. The position and size of the screen and internal projection parameters, i.e. b) and c) are static and can be calibrated and set-up beforehand.

For the actor's head positions a real-time head tracking is needed. This will be achieved using the video streams of the capture system and real-time processing of the IP-based system, as described above for the automated control of the PTZ camera head. The head position will then be streamed to a rendering module. The view-dependent rendering requires a specific setup of the projection. This might not be possible with specific rendering engines, but it is possible with rendering APIs like OpenGL and packages that give access to this level.

If a higher number of projectors is required, then the rendering of the virtual scene has to be distributed over several rendering PCs. This then requires the use of a rendering engine with a distributed scene graph. More details on the implementation of a view-dependent feedback system can be found in [1].

2. WP3 Performance Analysis

The objective of Work Package 3 is to develop the software components required to structure 3D video sequences of an actor into logical motion segments connected through a graph. WP3 will consider as input the acquisition data coming from the modules developed in WP2. This acquisition data includes: camera calibration information, image sequences from cameras and surface information in the form of sequences of independent meshes. The output of WP3 is globally consistent surface descriptions for each actor (i.e. with a single mesh-connectivity, as required for subsequent motion graph building) as well as appearance information. Although appearance is captured in view-dependent form of initial videos, a more compact, aligned texture representation of appearance is to be extracted from input videos to ease further rendering and transmission.

In order to organize data corresponding to different motions performed by different actors, data files will be structured in directory trees and described by xml meta files. In the following, the principles of such organization are given. Note that we want to keep this organization very flexible since it might significantly evolve in the course of the project.

2.1. Data format for shape, appearance and motion

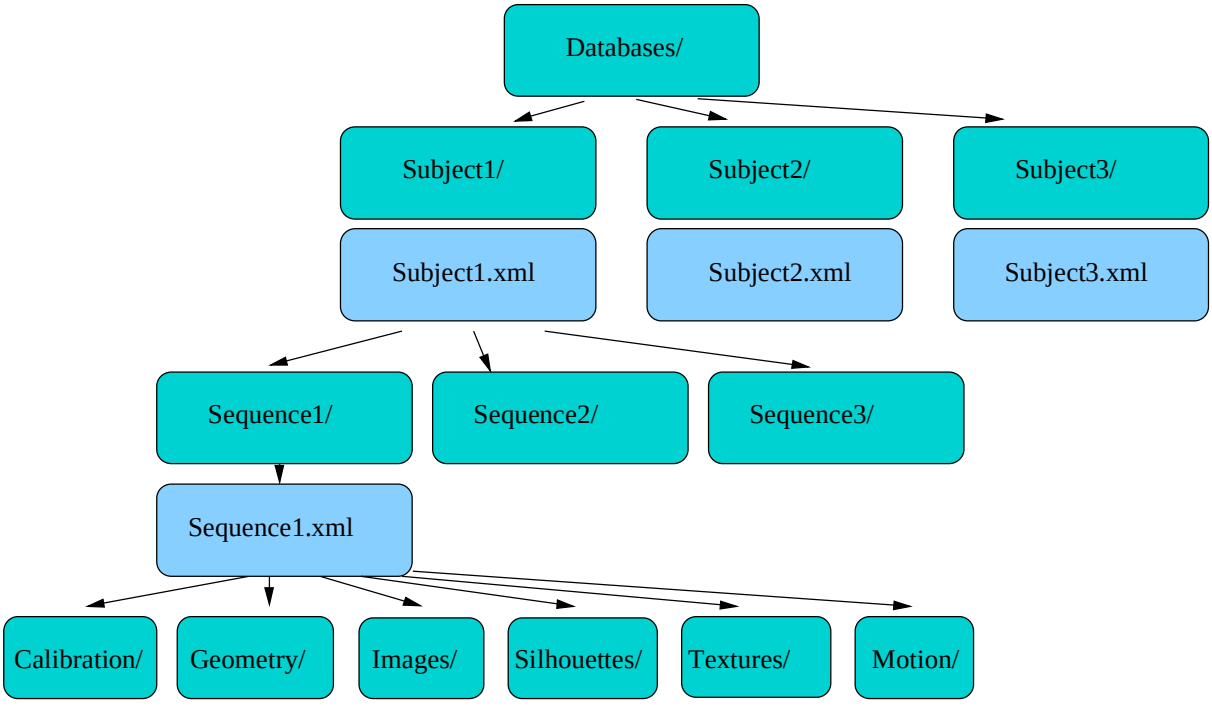

Illustration 4, general data file organization

2.1.1. Meta files

A database (e.g. subject1) is a group of temporally coherent sequences (i.e. all geometric descriptions have the same mesh-connectivity). A database is described by an xml meta-file. Shape, appearance and motion are then described per sequence using an xml meta-file for each sequence.

In the examples below, we assume that all sequences have similar structures and thus can be indexed with a sequence code over 2 digits. We also assume that a global timeframe encoded over 8 digits is available and can be used to index files. In addition, we assume that frames are homogeneous and thus do not need individual description in the xml file. This can be changed of course and a tag frame can be introduced to allow frame information to differ over time. On the other hand cameras can be of different types and thus a camera tag is defined.

Database file:

```
<?xml version="1.0" encoding="utf-8" ?>
<database version=”1.0” name=”Subject1” nbsequences=”..” >
<sequences  files=”/Sequence%2s/Sequence%2s.xml”/ >
<motiongraph file=”Subject1_Graph.xml”/>  described with WP4
</database>
```

Sequence file:

```
<?xml version="1.0" encoding="utf-8" ?>
<sequence version=”1.0” name=”walk1”  nbcam=”8” nbframe=”..” framerate=”..”>
 <setup>
  <camera id="0" type=”fixed” file="/Calibration/Cam00.txt"/>
  <camera id="1" type=”fixed” file="/Calibration/Cam01.txt"/>
  <camera id="2" type=”fixed” file="/Calibration/Cam02.txt"/>
  <camera id="3" type=”fixed” file="/Calibration/Cam03.txt"/>
  <camera id="4" type=”fixed” file="/Calibration/Cam04.txt"/>
  <camera id="5" type=”fixed” file="/Calibration/Cam05.txt"/>
  <camera id="6" type=”fixed” file="/Calibration/Cam06.txt"/>
  <camera id="7" type=”moving” files="/Calibration/Cam07/Cam07_%8t.txt"/>
 </setup>
 <geometry files="/Geometry/Surf%8t.obj"/>
 <motion files="/Motion/Surf%8t.bvh"/>
 <images cam="0" files="/Images/Cam00/Img00_%8t.png"/>
    .
    .
 <images cam="7" files="/Images/Cam07/Img07_%8t.png"/>
 <silhouettes cam="0" files="/Silhouettes/Cam00/Sil00_%8t.png"/>
    .
    .
 <silhouettes cam="7" files="/Silhouettes/Cam07/Sil07_%8t.png"/>
 <texture files="/Textures/Texture_%8t.png"/>
</sequence>
```

2.1.2. Camera description

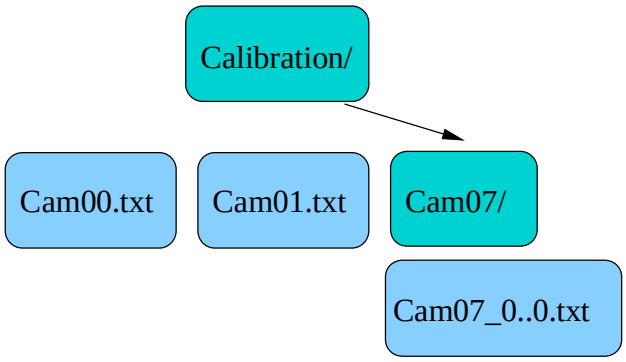

Illustration 5, calibration file organization

Cameras can be of two types, either fixed or moving. Fixed cameras are described in a single file. Moving cameras are described with several files indexed by a global time reference. This implies that usually one camera file per image frame is supplied.

2.1.3. Geometry

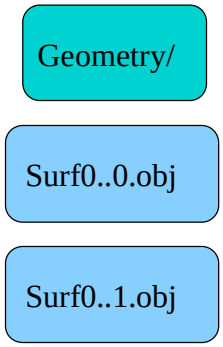

Illustration 6, geometry file organization

Geometry is represented in the OBJ format with vertices, normals and possibly texture coordinates if a texture map is estimated. Mesh connectivity can appear only in the first OBJ file of the sequence (in principle only once in the database). In the other OBJ files, only vertex locations can be given.

In order to estimate the geometry (i.e. meshes) in WP2, the silhouettes will be used to generate visual hulls, as explained in section 1.4. Visual hulls are approximate 3D models of the observed geometry. These models can be refined by considering the photometric information in the images

and optimizing them with respect to photoconsistency. A model is photoconsistent when it presents similar photometric properties in different images where it appears. We will use this principle and deform visual hull meshes so that their photoconsistencies are maximized (Illustration 7 shows an example of this principle).

Illustration 7, Example of a photoconsistent model: (left) an image of the original model; (middle) the visual hull obtained with several images of the model; (right) the photoconsistent model obtained by deforming the visual hull mesh

In order to estimate the appearance as a texture map, an approach needs to be developed. Both University of Surrey and INRIA are currently working on this topic.

2.1.4. Images

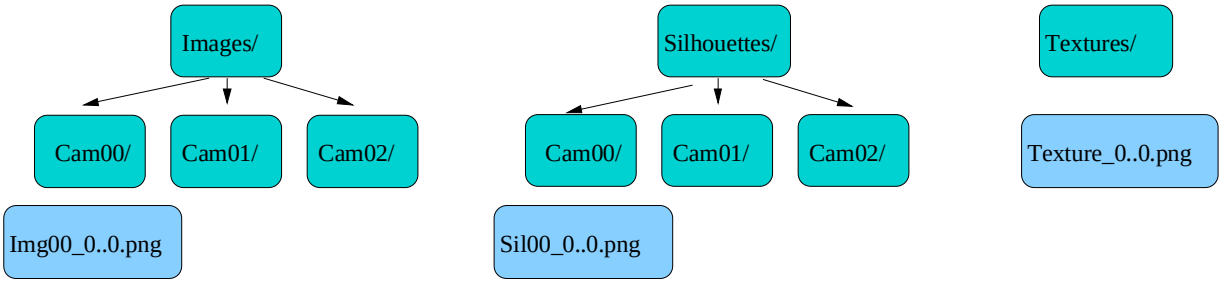

Illustration 8, image file organization

Images are stored in PNG format, as described above. A texture map for each frame can be stored as well, notably as output of appearance estimation algorithm. Further, silhouettes can also be stored with little effort as binary PNG files, since they can be useful in the further processing of the data.

2.1.5. Motion

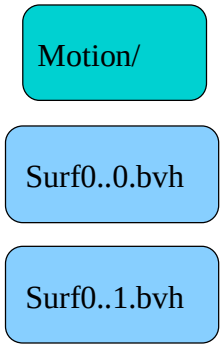

Illustration 9, image file organization

We are using the BVH (Biovision hierarchy) file format as it is a widespread and simple format. Other options such as FBX or COLLADA might be investigated during the second half of the project. These formats are more complex and difficult to implement. Hence, they might be supported if needed at a later stage in the project.

2.2. Streaming

For streaming geometry and appearance information, the OBJ format can be used as it allows several objects to be defined within a single file. A frame would then be composed of an OBJ file and one or more texture files. Binary formats should be preferred for that purpose. The OBJ binary format, i.e. .mod files, is proprietary and undocumented but a specific binary version can be built for RE@CT.

2.3. Interface for head and body representation

2.3.1. Head performance analysis

The goal of head footage analysis is to generate temporally and spatially consistent depth information for the head/face footage captured with the dedicated pan/tilt/zoom (PTZ) cameras during the shoot.

The separate acquisition of 3D reference models of the actors supports this step by providing detailed information about the head involved. However, pose and expression of the face in the PTZ footage deviates from the 3D model, which therefore serves primarily as a proxy for analysis.

First, a relationship between each PTZ video stream and the captured model is established. To this end, the individual PTZ video frames are matched with textured renderings of the 3D model using feature matching and image-based optimization techniques. Then the computed relation of the individual frames to the model is used together with the camera calibration to establish a consistent spatial relationship between synchronous frames of the different PTZ streams. Finally, each PTZ stream is tracked over time to provide temporal consistency.

The output of the analysis stage is:

- depth information for the region of each PTZ video frame showing the actor's head;
- inter-view correspondence information, relating the head region of each PTZ frame to its synchronous frames in neighbouring views; and
- temporal correspondence information, relating the head region of each PTZ frame to the next frame of the same video stream.

For motion-graph re-animation of the footage, semantic information on the head footage is required. This is to be achieved by semantically annotating the captured 3D models and propagating this information to the PTZ footage in the registration process. This may require some form of manual intervention for which appropriate tools are to be provided.

2.3.2. Fusion of body and head information

The required degree of fusion of head and body footage depends on the degree of interaction between head and body in the recorded performance. Some typical head shots such as dialogues or close-ups can be rendered from the head footage alone and thus do not require fusion. Other shots do require the integration of head and body footage.

It is important to distinguish between different aspects with respect to which parts of the body and head footage are to be fused:

- The captured 3D video of the body and that of the head must be brought consistently into a single world coordinate system, spatially as well as temporally. Otherwise detached heads may float in space when footage is rendered using both head and body-footage in the same video frame.
- For re-animation, the body pose information must be related to the head pose in order to constrain manipulations of the head pose to orientations which are consistent with the body pose. Also, precise depth information for the head pose is required if interactions between head and body occur, such as an actor scratching his head. Interaction like this will be addressed in the second half of the project.
- For rendering, smooth and consistent blending between body and head footage is required for scenes which fuse image content from both types of cameras. For blending, the head part of the 3D body video must be identified so it can be replaced by the head footage. Also the transition area must be specified.
- Finally, photometric differences between head- and body footage may have to be corrected, as they are captured with different cameras which may have different colour characteristics. Photometric corrections should be deferred to the rendering stage where other appearance corrections may occur.

The RE@CT capture process and data formats provide ample information which can be used for fusion. The following types of data will drive the fusion process:

- All cameras used in the capture process, head and body, are calibrated in a common coordinate frame and synchronized temporally. The calibration data is the most important information for relating head and body in the world coordinate system.
- The 3D video of the body does include head geometry, however in lower resolution than the footage from the dedicated head cameras. Still, this information can be used to initialize a registration process.
- The 3D video of the body also contains the head texture, again in lower resolution than the head footage but this information can still be exploited for fusion.
- The 3D video of both head and body are annotated semantically. Therefore, the location of the neck region and other relevant pose information is available for the fusion process.

3. WP4 Character Model and Animation Engine

3.1. Character Model

The RE@CT Character model is a hierarchy with motion graphs for both the face and body as shown in Illustration 10. Motion graphs link a set of motion classes as nodes to define transitions between different motions. Individual motion classes are based on one or more motion databases defined in the WP3 specification. The character model components can be summarized as follows:

Character Model: Defines the body and face motion graphs for a specific character together with the link between face and body motion.

Body/Face Motion Graph: Define the motion graph structure based on a set of motion classes linked by edges representing transitions between motion classes. Pre-computed information on transitions may also be stored (e.g. transition cost).

Motion Classes: Represent character motions as either a single (fixed) motion or a parameterized motion based on one or more motion sequences defined in a motion database. For parametric motions, the motion class defines the set of control parameters and the set of motion sequences from which the parameterisation is derived. Additional metadata for temporal alignment (synchronization) and similarity may also be included as required for parameterized motions.

Motion Database: Defines the data for the motion graph as a set of motion sequences together with alignment information. Individual motions may be surface, skeleton or image sequences. Detailed specification of the RE@CT motion database is given in WP3. Separate databases may be used for face/body sequences and different sequence types (surface/skeleton/image).

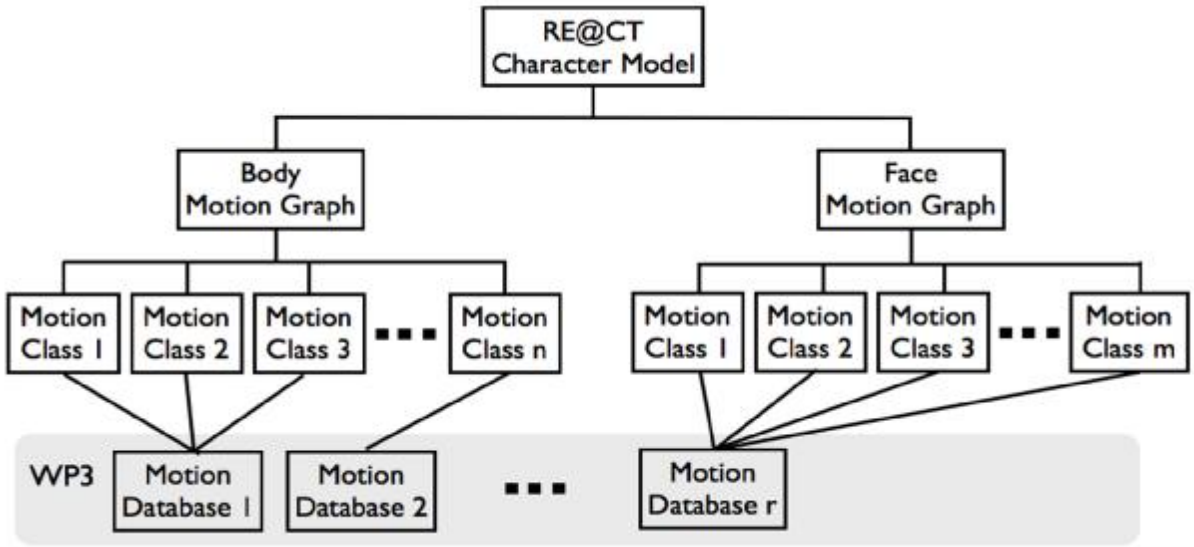

Illustration 10, RE@CT character model hierarchy

3.2. Non-parametric vs. Parametric Motion Graphs

The motion graph is the basis for creating animated characters in RE@CT, and a character model may comprise motion graphs for both face and body animation. Motion graphs enable the creation of new animations by concatenating fragments of motion capture from one or more captured frame sequences within a motion database, as specified in WP3. In the classical formulation of the motion graph, fragments are played back unchanged; we refer to this here as a non-parametric motion graph. In the parametric form of the motion graph, two synchronized fragments may be blended by some relative weighting to create novel fragments. The proposed animation engine unifies both types of motion graph and therefore requires a data format to represent both parametric and non-parametric motions.

In a motion graph each motion fragment is represented as a node, and possible transitions between fragments are specified by edges. We refer to nodes as motion classes, since in the parametric case, a fragment may be capable of expressing a variety of parameterized motions. Illustration 11 shows that parametric and non-parametric motion classes may be combined in the motion graph.

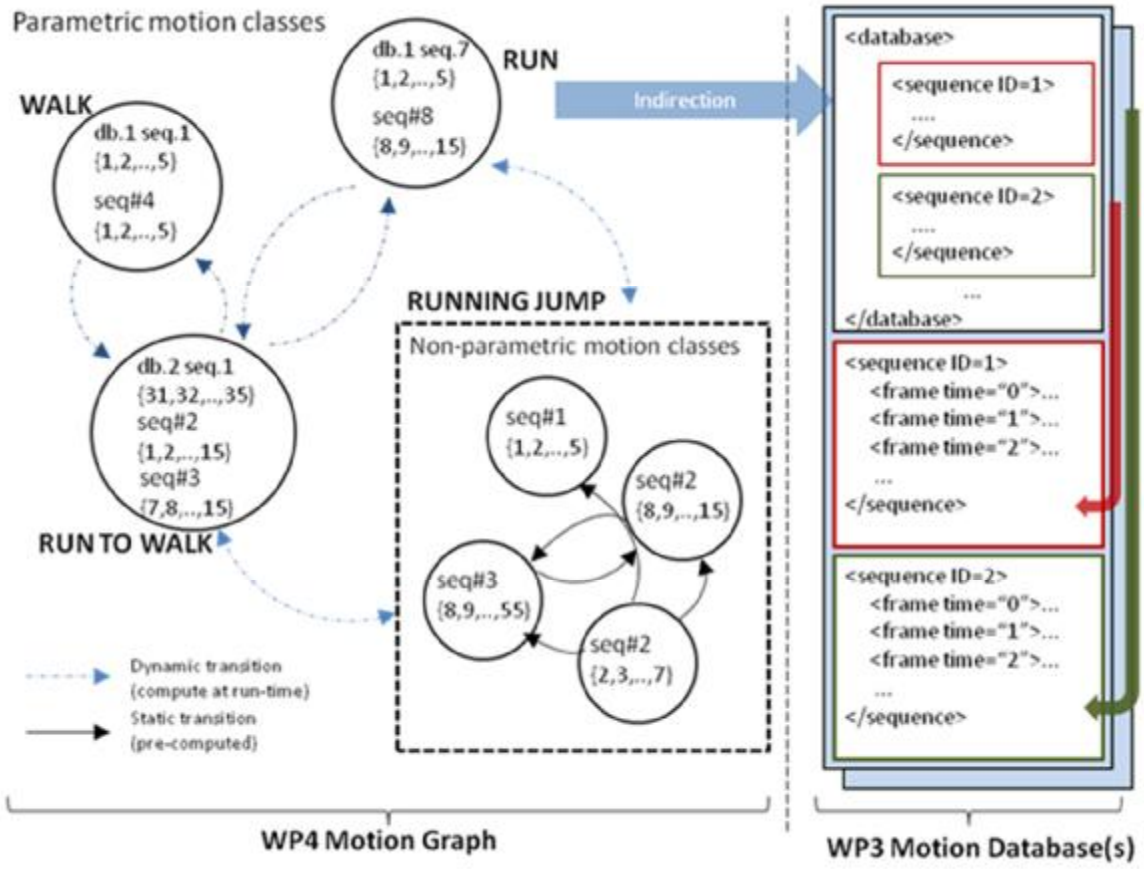

Illustration 11, a motion graph comprising part of the character model (e.g. the full body animation). The graph may consist of parametric and non-parametric motion classes, and references the motion database specified in WP3 using sequence and frame number.

A further implication of the two types of motion class is that whereas the possible transition paths between motion fragments may be pre-computed in a non-parametric motion graph, the number of possible transitions and blends in a parametric motion graph cannot be computed for sequences of

practical length. This consideration is reflected both in the data specification document, and the specification of the animation engine.

3.3. XML format for RE@CT Motion Graphs

Motion graphs are created from one or more sequences of motion capture, which comprise frames of mesh, skeleton, texture and other data. The data format of these sequences is specified in WP3. Here we assume only that:

- sequences used in the motion graph originate from one of more databases e.g. 'subject1walk', 'subject1run', as specified in the 'name' field of the <database> tag.
- sequences have a unique identifier e.g. 'walk1', e.g. as specified in the 'seq' attribute of the <sequence> tag in the database xml file of WP3. This identifier is unique within the namespace of its own database.
- sequences comprise T frames with integer numbering [0,t-1] as per the "time" attribute on sequence XML files defined in WP3.

3.3.1. Top-level structure of the XML file

The basic format for the motion graph XML file is:

```
<?xml version="1.0" encoding="utf-8" ?>
<motionGraph version="1">

      <motionClass parametric="no">
            …
      </motionClass>

      <transition>
            …
      </transition>

</motionGraph>
```

One or more <motionClass>

Zero or more <transition>

The version of the motion graph schema is 1. This will enable future evolution of the schema.

The type of the motion class may be either non-parametric or parametric, as indicated by the "parametric" attribute. In the case that the graph is entirely parametric there are no <transition> elements, because transitions are computed at run-time during the animation. In the case of motion graphs containing non-parametric motion classes, which contain pre-computed transitions (but transitions might also be computed at run-time) there may or may not be <transition> elements.

3.3.2. Motion Classes

Each node in a RE@CT motion graph represents a motion fragment with specific start and end time index; i.e. a motion class. Each motion class is also assigned a unique integer identifier (ID), which implies no ordering and is for reference only.

In the case of a non-parametric motion class, there is reference only to a single motion capture sequence. In the case of a parametric motion class the node refers to one or more synchronised motion capture sequences, which may be blended together.

The sequences may be of differing duration in frames, as indicated by the start and end attributes of their respective <sequence> element (see below). The sequences are referred to by database name and sequence ID, as defined in the database XML file of WP3. For the purposes of local reference within the <parameter> element (described shortly), each sequence is also assigned a handle ("alias") unique within the namespace of the local motionClass, for example "seq1".

The format of the node section of the XML is

```
<motionClass parametric="yes" ID='1234'>
      <sequence start='0' end='5'>
            <database>x</database>
            <ID>5678</ID>
            <alias>seq1</alias>
            <frame index='0'>
                  <rotation>
                        <row> 0.234 0.567 0.89 </row>
                        <row> 0.234 0.567 0.89 </row>
                        <row> 0.234 0.567 0.89 </row>
                  <translation>
                        <row> 0.234 0.567 0.89 </row>
                  </translation>
            </frame>
            …

      </sequence>

      <parameter name="A1" type="linear">
            <sequences>
                  <alias>seq1</alias>
                  <alias>seq2</alias>
                  …
            </sequences>
            <synchronization>
                  <mapping alias='seq1' units='seconds'>
                        <source>0.1</source>
                        <dest>0.2</source>
                  </mapping>
                  …
            </synchronization>
      </parameter>
</motionClass>
```

N sequence elements

frame elements repeat for all frames from start to end

N-1 parameter elements (only occur in parametric case)

Optional synchroniza-tion data for aligning sequences

Sequences are identified by an ID and database name (see WP3) and comprise multiple frames each of which is numbered according to the time index (frame offset from start) as defined in WP3. Each frame is encoded as per the schema above, with the rotation and translation of the model relative to the origin (as specified in the coordinate system of the camera calibration, defined in WP3). The models themselves are assumed to be stored within a local coordinate system relative to the centroid of the model, as defined in WP3.

The <parameter> elements occur only in the case of a parametric motion class (i.e. when a node has more than 1 sequence) and define a naming convention for blending between certain sequences. So, in the example above a scalar parameter named "A1" blends between two sequences within the aliases "seq1" and "seq2" (the latter is not depicted in the example above). The type of the blend in

this example is “linear”, however this name is a placeholder and blend types for parametric motion classes will be defined in later technical WP4 deliverables. Any number of sequences may be blended by a single parameter.

Within the <parameter> element there may optionally be stored synchronization data to align temporally a pair of sequences blended by that parameter. This temporal alignment is specified in a piecewise linear mapping using the <mapping> element. Points of similarity are specified between a pair of sequences using their locally defined alias. These points may be identified in units of seconds offset from the beginning of the sequence, as denoted by the units attribute. In the future, additional units may be defined within this specification. The temporal alignment data is pre-computed to enable run-time blending.

3.3.3. Pre-computed Transitions

In addition to <motionClass> within the top level <motionGraph> element, there may be a single element <transitions>. This stores pre-computed edges in the motion graph. Because only non-parametric motion classes can pre-compute transitions, the <transitions> element may not be present e.g. in entirely parametric motion graphs.

The transitions element stores a set of <edge> elements each describing a weighted directed edge between nodes in the motion graph. The format is

```
<transitions>
      <edge>
            <connectivity>
                  <source>1234</source>
                  <target>5678</source>
            </connectivity>
            <weights>
                  <value label=”colour”>0.5</value>
                  <value label=”shape”>0.3</value>
                  …
            </weights>
      </edge>

      …

</transition>
```

Zero or more weights per edge

Any number of edges may be coded between nodes

There may be any number of edges specified, and the integer references within <source> and <target> are indirections to the <node> ID attribute in section 3.3.

Each edge may be annotated with weights. In the above example the weights are a vector quantity of dimension 2, the first component relating to colour similarity the second relating to shape similarity. These labels are arbitrary and in practice similarity between motion classes may be computed in any number of ways.

3.4. RE@CT Character Animation Engine

The RE@CT animation engine operates between the low-level game engine (for example Unity or some other OpenGL based rendering framework), and the high-level planning and scenario logic governing the application control (Illustration 12).

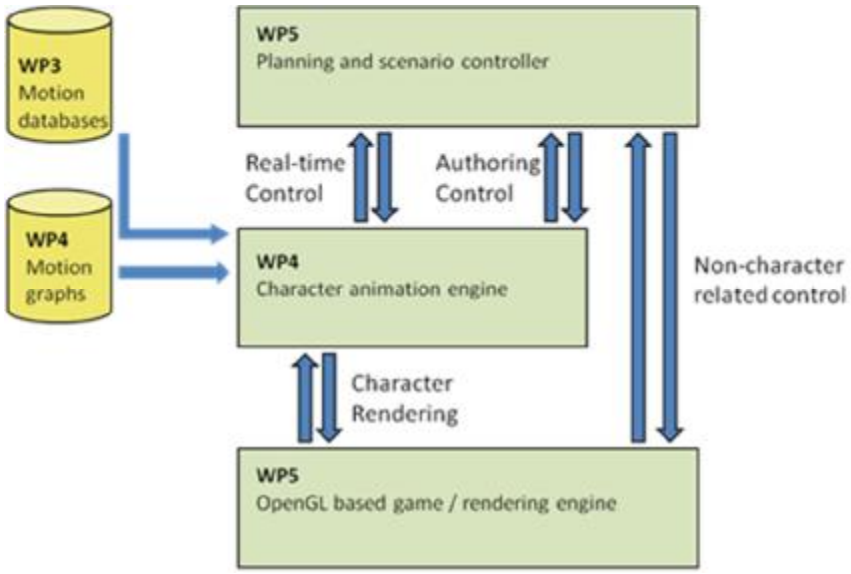

Illustration 12, interaction of the character animation engine and other RE@CT subsystems.

The character animation engine (CAE) maintains state of all the characters currently being animated. There are three modes of interaction between the CAE and the other subsystems:

1. Character Rendering. The CAE is called upon during the render/display loop of the games engine to paint (i.e. upload to the GPU the necessary geometry/texture) the animated characters at their current position. This geometry is determined by the internal state of animated characters maintained within the CAE.

2. Interactive Control of Character Animation. The CAE is called upon by the subsystems responsible for high level planning/control of the animation to initiate some form of movement. This would be specified at a low-level e.g. “start character A walking with a particular velocity”. A more complex alternative might be “start character A walking with a particular velocity which will transition into a run in X seconds with a new velocity”. Parameters may be specified on the motion; for example, speed or stylistic constraints on the motion. However it will not be possible to specify requests at a higher level to the CAE e.g. “Character A should walk to a particular position, avoiding particular obstacles” will not be possible. This kind of high level constrained planning will be undertaken by WP5 which will then issue low-level “course correction” commands interactively to the CAE to steer the character. Further to the full-body character animation examples given here, similar kinds of low-level direction may be given to the CAE to the facial animation, which will also be specified using a motion graph.

3. Authoring (direct scripting) of Character Animation. The CAE may be called upon to manipulate the Character mesh directly, for example to script a particular sequence of motions such as reaching for a target. In this case the motion graphs of the CAE are not used; rather, the CAE simply offers via

pass-through the direct control of the mesh to the planning/control module of WP5. Thus the application has the ability manually to script actions, carefully designed offline / a priori, in addition to requesting control of the mesh via method (2) interactively.

3.4.1. Outline of the Character Animation Engine API

The following outline API specification will be used to implement the above three categories of functionality.

(1) Character Rendering

draw(*characterID*) – the engine will be called upon to draw characters currently being animated based on the internal state of the animation engine.

getMesh(*characterID*) – the engine will return the geometry of characters currently being animated e.g. for purposes of collision detection and planning.

(2) Character Animation

animate(*characterID, motionClass, motionParameters*) – the engine is called upon to begin animating (or switch animation of) the character to a new motion class e.g. "run". Parameters may be specified such as velocity, style, and desired number of seconds in which to achieve the transition to the new motion class.

animateFace(*characterID, motionClass, motionParameters*) – the engine is called upon to animate the face to a new position, in a similar manner to *animate.* However only the facial part of the mesh will be affected, and this will be independently controlled from the full body animation.

getState(*characterID*) – return the current motionClass and related parameterization information for a particular animated character's body.

getFaceState(*characterID*) – return the current motionClass and related parameterization information for a particular animated character's face.

constrain(*characterID, constraintInformation*) – called to impose low-level constraints on the full body character animation, for example to communicate foot-floor constraints. Note that it will not be possible to specify physical constraints such as trajectories, obstacles, and the like – it is assumed that the WP5 controller has taken such concerns into account when planning its calls to *animate()*.

(3) Author/Script Animation

setMesh(*characterID, meshPositioningData, timingData*) – particular vertices of the mesh may be directly manipulated by the caller to script a sequence of moves that may not occur within the motion capture data, for example to orchestrate a series of movements that will precisely grasp an object. The sequencing of these movements would be achieved by the caller i.e. WP5.

setMeshBySkeleton*(characterID, skeletonData)* – use a skeleton to directly drive the animation of a mesh for the same purposes as *setMesh().*

4. WP5 Immersive and Interactive Platform

This work package will develop the interactive components of the project, based on a widely used graphical 3D rendering engine, Unity 3D. This will be used in both the professional interactive authoring tools as well as the end-user interface, since both will be based on the same underlying rendering engine. Authors of interactive content need direct feedback from the virtual environment and the ability to manipulate the content interactively as part of the authoring process. Conversely, the end-user will only use the animation and rendering part of the platform.

WP5 will integrate the algorithms and methods developed in previous work packages and will add a graphical user interface. The RE@CT animation engine from WP4 providing control of actor performance will be integrated with the 3D real-time rendering engine in order to display and manipulate realistic animations. WP5 may also enable networked streaming of interactive content by developing compact representations of the data for efficient delivery.

4.1. Rendering Engine and messaging system specification

4.1.1. General requirements

The rendering engine is capable of rendering 3D data in real-time and has two purposes: it is used for the user-facing end product and also as a module in the authoring tools to provide an intuitive Graphical User Interface with real-time feedback.

There are different choices for a software platform to implement the engine: a low level rendering API such as OpenGL technology is not adequate, as too many functionalities found in games engines are missing. However, currently the rendering approaches used or image-based appearance rendering requires low-level and advanced OpenGL functionality. These features are usually not integrated into most of the available high-level rendering APIs or games engines. After testing several of the different available 3D rendering engines, we decided to use Unity3D for the reasons described below.

The Unity3D platform has an integrated graphical authoring tool and allows fast development of applications + content. An editor allows the design of the GUI and provides simple tools for editing scripts. Unity3D includes an optimized 3D and 2D rendering tool. Unity3D is a commercial product, and it lacks some elements of flexibility, such as the ability to tweak its low-level functionalities. However, Unity3D provides a plug-in system that allows developers to write their own libraries and integrate them into a Unity Application. Unity3D is capable of integrating external OpenGL code that runs in the Unity3D environment. Further, Unity provides support for a variety of platforms, like Windows, MacOS X, Android, iOS, Wii... This fulfils the requirements regarding the targets for our demonstrators.

As several tests conducted by Artefacto indicated, Unity3D is compatible with all the technologies being developed during the project. In addition to rendering, it will have to dispatch and get information, data, and commands from different modules in charge of different aspects of the immersive and interactive system. To ease the communication between the different modules, we will set-up a communication system based on an API described in the section above.

With the specifications of the platform evolving over time, it appears that no network capabilities will be required, so the original specification for network communication is no longer relevant.

4.1.2. 3D engine specifications

The most important aspect being tackled by the project is the flexibility for integrating external code. This should allow all partners to develop their own library and provided that all comply with default interfaces being predefined, the integration will be eased. Unity3D complies with this requirement.

Partners will define the basis of the interfaces that will be used during the project to allow for library integration and data exchange. This will result in headers and precompiled libraries which will be available for the project partners to share.

(1) OpenGL Capabilities

During the initial phase of the project, partners collected information about the technologies currently used by individual partners, and gathered a set of minimal requirements the rendering engine should comply with, with regards to the low level 3D functionalities being embedded.

A non-exhaustive list of required features includes:

- Draw a mesh
- Draw textures
- Use of Frame Buffer Objects
- Use of Multiple Resolution Targets,
- Use of Vertex Buffer Objects
- ...

All the latest 3D engines should comply with these. However, the difficulty will be to integrate external code that uses those low-level OpenGL functionalities into the main rendering loop of the engine.

The RE@CT project partners will also focus on the quality of the geometry being rendered. For demonstration purposes and in a later phase, partners may integrate texturing of the animated objects.

(2) GUI capabilities

Apart from the OpenGL aspects, the 3D engine should allow for an easy and friendly way of designing graphical user interfaces.

Buttons, Sliders, Toggles, Checkbox, gesture interpretation, external event handling must be part of the default GUIs made available by the engine. This will make it possible to design and adapt the interfaces easily as the project evolves.

(3) Minimum requirements

Two 3D rendering engines had initially been identified, which comply with our initial requirements: OGRE 3D and Unity3D. The project has now made the decision to use UNITY 3D as the platform to be used for the authoring tool and the end-user interface.

OpenGL integration

Unity3D has a plug-in system. This will need further investigations to see if it is possible to call low level OpenGL rendering functions during the main rendering loop. If this test is successful, the RE@CT partners propose to aim to make their library compatible with the Unity3D engine. One risk might be over the rendering performance. Regarding the project objectives, we will consider mesh rendering but not tackle the texturing of the mesh.

User-friendly external code integration

One constraint is to allow all partners to develop extensions to the system for their respective tasks and ease the integration of components for the demonstrator. Unity3D provides this facility. It relies on a plug-in system that is sufficiently flexible to allow for the incorporation of the required low-level functionalities.

Unity3D has been tested and OpenGL calls can be performed within its standard rendering loop.Given the cross platform capabilities of Unity3D, this engine was chosen as the common platform to develop both the End User application and the interactive authoring tool in the RE@CT project.

Provide a test-tool for testing the integration

Once the message protocols and the interfaces between the rendering and the animation engine have been defined, a test platform will be provided to allow partners to test their code as and when needed. This will be provided as a binary along with a sample code for integration. This binary will be available for the Windows/MacOSX or Linux Operating systems.

4.1.3. Reading mesh sequences

With the tests performed during the first half of the project, we set up an initial platform both on a standard PC and on a mobile device. We integrated state of the art Free Viewpoint Rendering techniques available from the project partners and we concluded that these techniques may not be suitable for mobile platforms. Free viewpoint rendering requires that the different camera views are loaded and rendered using depth layers. This has been shown to be extremely efficient [3], however to allow for real-time display of the mesh sequences, data must be preloaded, and within the context of an interactive reanimation scheme, this is not possible. We have therefore modified the data transmission scheme. As an output, University of Surrey provides an initial geometry (the mesh) and single textures that can be directly applied to the geometry within the 3D Engine. Provided textures must still be of a reasonable size, so that reading from the hard-drive is possible within 1/25 seconds.

4.1.4. Interfaces and message passing system

As discussed above, the RE@CT project aims to provide a real-time animation system. The RE@CT architecture can be split into two categories of data processing: real-time and offline (non-real-time), as shown in Illustration 13.

The interactive and immersive parts are the animation engine, the authoring tools and the streaming to the end-user.

As data processing in RE@CT is computationally demanding, the project will look into whether distributed computation is required. If this is the case, programming interfaces as well as a message passing system for communication between the components distributed in the grid will need to be defined. For reasons of simplicity, the grid should not contain more than three or four machines. Network computation has the additional advantage that heterogeneous architectures can be used during the development of the project. For example, the algorithms of some partners can run on a Linux operating system while others can use Windows.

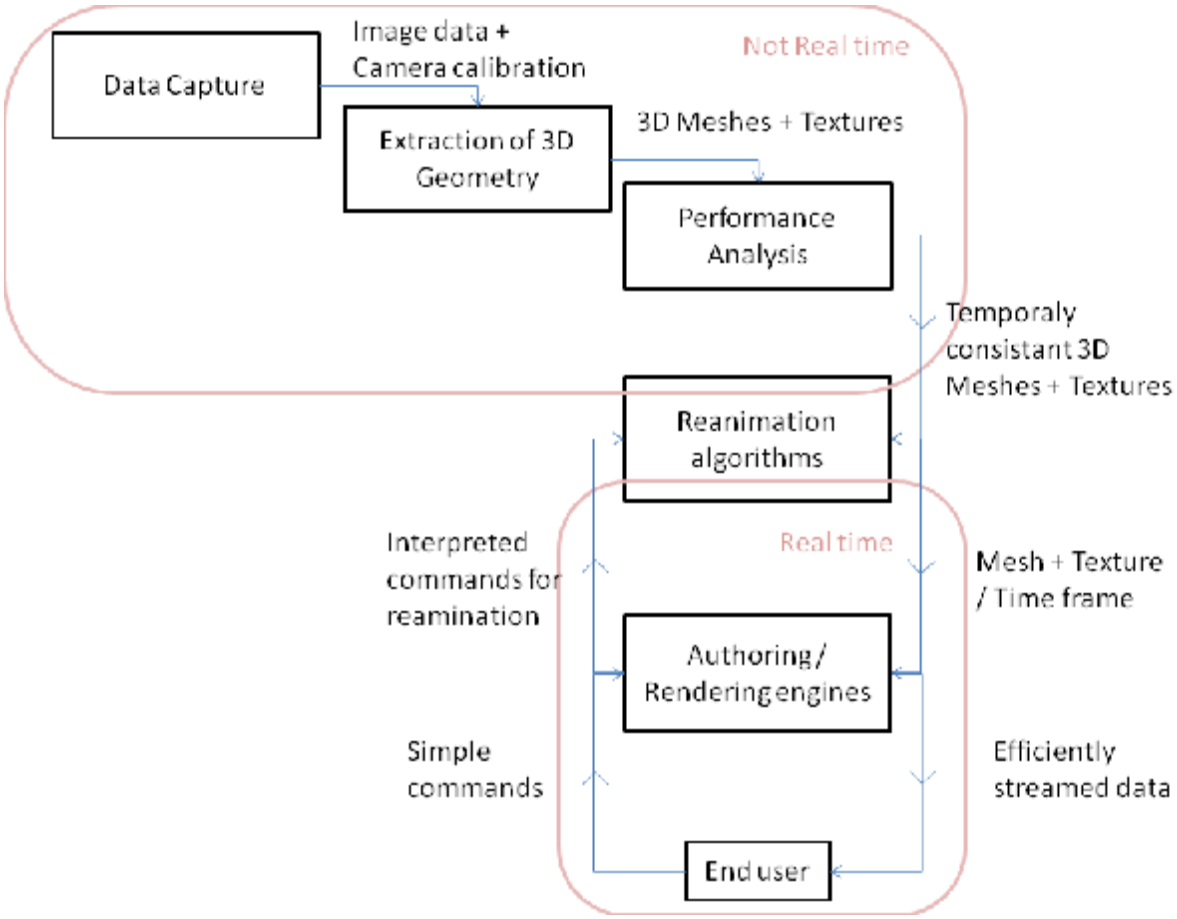

Illustration 13, global scheme for RE@CT message passing flow

4.2. Compact representations for efficient delivery

RE@CT's high quality 4D actor performance capture and motion graph representation will potentially result in large amounts of data that will need to be accessed by the animation engine to render new images. In order to make the rendered output available to the end user, two different approaches for delivery will be investigated. One approach to provide end-to-end solutions for interactive rendering of highly complex computer graphics scenes to a wide variety of end devices is rendering in the cloud. A dedicated server interactively renders the scene based on user input. The output is captured, encoded as video and streamed to the client. The challenges of this method are the reduction of delay in the streaming and the encoding complexity. An alternative solution is the rendering on the client side which requires the transmission of the data over the network. While the development of a new codec for the novel data representation is beyond the scope of the RE@CT project, the data structure will be optimized in order to enable random access and efficient view-dependent delivery. The two approaches are described below in more detail.

4.2.1. Interactive streaming of rendered output

In the video-based streaming method, most of the responsibilities are being shifted to the server-side infrastructure, as application execution, rendering of the performance, video encoding and streaming will be centralised there. For real-time streaming of interactive content, the system components need to fulfil specific technical requirements in order to guarantee a good overall user experience.

Server Side Environment: The basic requirement for streaming rendered graphics as video is to gain access to the graphical context and the frame buffer of the rendering application. For this purpose either the application directly supports that access or the graphics API used for drawing must be

intercepted. To ensure minimum latency, both the graphics rendering and the video encoding should be hosted on the same physical machine. The server must provide enough resources in terms of CPU, GPU and RAM capacity to ensure real-time processing at all times. This may also require constant monitoring of the system load dynamically to limit the frame rate of the rendering application and/or the frame buffer capturing.

Video Encoding: The main requirements for video coding in this scenario are real-time processing and very low delay. Therefore encoder features with high coding complexity, as well as features that introduce out-of-order temporal dependencies (e.g. bi-directional prediction, look-ahead analysis) into the encoding process, should not be used. Encoder parallelization may be exploited, if content with high resolution and/or motion complexity cannot be encoded in real-time on a single CPU-core. If the streaming layer does not handle transmission errors such as packet loss, the video should be encoded to allow for error recovery. To prevent the introduction of additional latencies into the networking layer, the encoder must also respect the bandwidth constraints of the transmission channel. The use of rate-control and continuous Intra-updates should be considered to shape the bandwidth of the resulting bit-stream. Within the project, we will focus on H264/AVC as codec since it provides high coding efficiency and decoding on a large variety of end devices.

Network Transmission: On the streaming layer, specialized protocols with low delay and protocol overhead, such as RTP over UDP, should be chosen for transmission. Protocols that require large fixed-sized data packets or continuous handshaking are typically not usable. In controlled network environments, the use of Quality of Service techniques should be considered to improve the reliability of the streaming.

End Device Capabilities: End devices must be capable of decoding H264/AVC video at a given encoder profile and level, typically by utilizing a built-in H264/AVC decoder chip or a programmable digital signal processor.

4.2.2. Scalable representation for efficient delivery

In contrast to rendering realistic animations in the network, local processing on the end-device requires streaming or downloading large data structures, raising the need for efficient representation. The data representation to be encoded, however, consists of a number of heterogeneous elements requiring different codecs. Appearance data is related to the multi-view video sequences captured by the camera arrays. Additionally, surface representations of the actors by means of animated polygonal meshes need to be addressed as well as skeletal motion in the motion graph. For individual data elements, codecs for compression already exist while the specification of a full coding and transmission system during the development of the data format is not targeted. Instead, a new scalable data representation is specified, that fulfils the requirements of random access to partial data necessary for view dependent rendering. This type of access differs significantly from the linear stream of temporal data such as video or audio and needs investigation.

RE@CT will develop methods for a temporally-coherent representation of both the geometry and appearance for actor performance capture. Scalable representations will be exploited in RE@CT as an efficient means of reducing the bandwidth by decoding only partial data based on the required temporal or display resolution. The concept of surface motion graphs requires more sophisticated access to the compressed data for interactive access with temporal jumps and loops between 3D video segments. Similarly, spatial dependencies need to be modified in order to enable partial view dependent streaming of the multi-view data. RE@CT will introduce novel efficient prediction structures for non-sequential access optimized to interactive media.

References

[1] O. Grau, T. Pullen, G. Thomas, "A combined studio production system for 3D capturing of live action and immersive actor feedback", IEEE Tr. on Systems and Circuits for Video Technology. March 2004 .

[2] Wikipedia, .obj file format, http://en.wikipedia.org/wiki/Wavefront_.obj_file

[3] J. Imber, M. Volino, A. Hilton, J.-Y. Guillemaut, S. Fenney, "Free-Viewpoint Video Rendering for Mobile Devices", MIRAGE, 2013