Free-viewpoint Immersive Networked Experience

D6.3 Free-viewpoint video player

Project ref. no.	ICT-FP7-248020
Project acronym	FINE
Start date of project (dur.)	1 April, 2012 (36 months)
Document due Date :	1, 8, 2012
Actual date of delivery	30, 7, 2012
Leader of this deliverable	BIT
Internal Reviewer	Sammy Rogmans, sammy.rogmans@uhasselt.be
Reply to	Peter.Schickel@bitmanagement.de,
Document status	Final

Deliverable Identification Sheet

Project ref. no.	ICT-FP7-248020
Project acronym	FINE
Project full title	Free-viewpoint Immersive Networked Experience
Document name	Free viewpoint video player, first version
Security (distribution level)	Public(PU)
Contractual date of delivery	Month 12, 01.08.2012
Actual date of delivery	Month 12, 30.07.2012
Deliverable number	D6.3
Deliverable name	Free viewpoint video player, second version
Type	Report
Status & version	Final version
Number of pages	13
WP / Task responsible	WP6/BIT
Other contributors	
Author(s)	Peter Schickel, Holger Grahn
EC Project Officer	Manuel Carvalhosa
Abstract	This document outlines the second free viewpoint video player as developed in WP6
Keywords	visualization, multi-view
Sent to peer reviewer	*28.03.2012*
Peer review completed	
Circulated to partners	Via plone
Read by partners	Via plone
Mgt. Board approval	To be approved at the next SB meeting

Table of contents

1 Introduction

The purpose of this document is to describe the second version of the free-viewpoint video player prototype as in WP6.

For this deliverable Bitmanagement has developed the second version of the fine free viewpoint video player according to WP 6 Task 1 as a multi-view compliant player that enables both photorealistic as well as 3D rendering. APIs to plug-into Bitmanagement's BS Contact player, new media decoders and network interfaces for fine particular applications, e.g. video streaming have been developed in order to cover the needs from other related work packages, especially WP6 and WP4. This has been done according to the requirements described in deliverable 2.2. Test data for video mesh generation from UPF and gesture reconstruction data from KTH has been visualized as well as a test system for computer graphics rendering has been developed. The following describes the results in rendering the diverse visualization needs of the project. The rendered output is visualized and described in brief for video depth maps, triangulation and the visualization of artificial computer graphic soccer avatars as well as the visualization of free viewpoint meshes from WP 4. The results are positive as the diverse rendering needs of the project have been addressed and implemented.

2 Visualization of video depth maps and 3D triangulated meshes from video

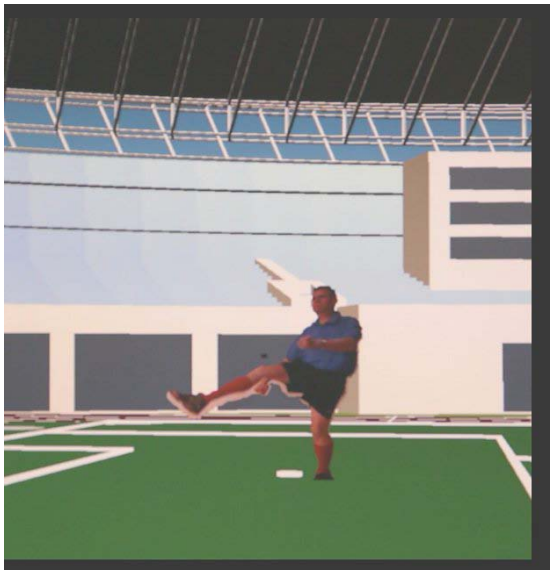

Figure 2.1: Visualization of a 3D mesh from height-field data

In addition to the ROI coded video sequences of deliverable 6.1 that resulted in a flat 2D video with alpha channel border cut inserted in a 3D environment Bitmanagement has developed a capability in the FINE player to visualize truly 3D avatars based on video textures and depth maps. From the live video feed and live depth map stream a triangulated mesh is constructed in real-time from a X3D height-field. The mesh shows 3D depth as well as accurate textures and is inserted in the 3D soccer scene as an object. The positions of the resulting 3D video avatar can be changed according to the real player positions of the soccer players supplied by the FINE multi-user server.

However the 3D mesh depends on the number of depth cameras viewing the scene. If one camera is used only as above the maximum viewpoint change may be a turn of about 30 degrees as an oval shape height-field (figure 2.1). The viewing experience can be enhanced if the oval shape height-field is inserted as a billboard object always turning towards the viewing position. The positions of the resulting 3D video avatar can be changed according to the real player positions of the soccer players supplied by the FINE multi-user server.

Figure 2.2: Shaded 3D mesh with colour vertex information

Meshes constructed from many cameras exhibit a round object that can be seen as avatars from any angle, such as the above 3D mesh supplied by UPF from WP4 to Bitmanagement. However this kind of data constructed from video feed frames result in very large 3D scenes for single images and is difficult to visualize in realtime with standard methods. Bitmanagement has developed a possibility to optimize the scene data as well as the rendering speed for real-time replay with intelligent geometry and texture coding from WP 5 based on the BS Contact 3D engine. Please see deliverable 5.4 for details.

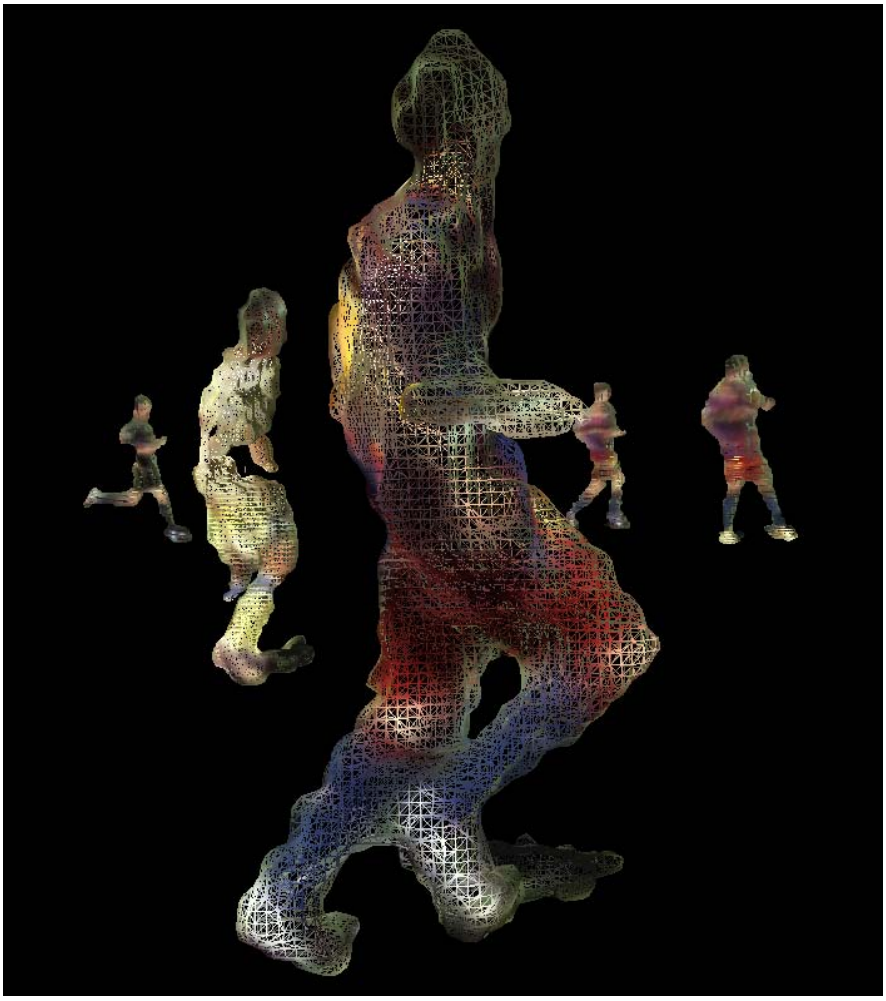

Figure 2.3: Highly detailed 3D mesh shape from one video frame

3 Visualization of gestures in graphic soccer avatars with the H-ANIME Standard

Overview H-Anime

H-Anime (Human Animation, [1]) is a standardized way for representing virtual humans. Virtual humans or Humanoids are often also referred to as Avatars. The H-Anime standard has advantages with regards to the simplicity of the animation of body parts of a virtual human being and the segmentation from the animation sequence and the modelling. An Avatar in H-Anime therefore consists of a set of segments (e.g. feet and hands) that are interconnected with joints (elbows, ankles). The angles of the joints can be changed according to the animation at runtime. The result is an animated avatar that moves the body parts according to an animation string made from angles.

X3D and HTML5

The implementation of the animated players in X3D with the H-Anim standard constitutes a transition path towards emerging technologies such as HTML 5 and WebGL rendering. First implementation with X3D in WebGL are using 3D renderers written in Javascript. The rendering speed of these Javascript implementations are still slow today compared to native C++ rendering. However there is a strong initiative to include X3D content into HTML 5, as the W3C consortium states that
"*Embedding 3D imagery into XHTML documents is the domain of X3D, or technologies based on X3D that are namespace-aware*." ([2])

Therefore the soccer avatar content in the Fine project represented in X3D is future prove and will inherit a long life time and compatibility from standardized formats.

KTH has supplied Bitmanagement with captured and reconstructed gesture data from their FINE research. The FINE player has been amended to support these gesture data. After modelling the avatar the data from optical tracking had to be translated in the ISO H-ANIME format to match the gestures with the body parts. The resulting scene has been ported into X3D binary for fast replay. The animation of the scene has been done by dynamic scripting in the X3D scene according to the H-Anime 1.1 Joint hierarchy. The X3D scene is now lively animated and can be used for simulating advanced running and jumping behaviour in the avatar.

```
PROTO Humanoid [
  eventIn       SFFloat    set_fraction
  eventIn       MFNode     set_joints
  eventIn       MFNode     set_skeleton
  eventIn       SFNode     set_skinCoord
  field         SFVec3f    bboxCenter          0 0 0
  field         SFVec3f    bboxSize            -1 -1 -1
  field         MFNode     joints              [ ]
  field         MFNode     skeleton            [ ]
  field         SFNode     skinCoord           NULL
  exposedField  SFVec3f    center              0 0 0
  exposedField  MFString   info                [ ]
  exposedField  SFString   name                ""
  exposedField  SFRotation rotation            0 0 1 0
  exposedField  SFVec3f    scale               1 1 1
  exposedField  SFRotation scaleOrientation    0 0 1 0
  exposedField  MFNode     segments            [ ]
  exposedField  MFNode     sites               [ ]
  exposedField  MFNode     skin                [ ]
  exposedField  SFNode     skinNormal          NULL
  exposedField  SFVec3f    translation         0 0 0
  exposedField  SFString   version             "2.0"
  exposedField  MFNode     viewpoints          [ ]
  eventOut      MFNode     joints_changed
  eventOut      MFNode     skeleton_changed
  eventOut      SFNode     skinCoord_changed
]
```

Figure 3.1: H-ANIME 1.1 compliant Humanoid proto

```
// Transforms the vertices related to a joint
        function Transform() {
          // Make sure that this is a joint
          var j;
          for (j=0; ((j<iNumJoints) && (joints[j].name != nextjoint.name)); j++);

          // If it is, we process the data
          if (j<iNumJoints) {
            // Read in current next joint
            var currentJoint = nextjoint;
            // Read in current matrix
            var currentMatrix = new VrmlMatrix();
            currentMatrix.setTransform(translation,
                                       rotation,
                                       scale,
                                       new SFRotation(1,0,0,0),
                                       new SFVec3f(0,0,0));
            // Create matrix corresponding to this joints transform
            var newMatrix = new VrmlMatrix();
            newMatrix.setTransform(currentJoint.translation,
                                   currentJoint.rotation,
                                   currentJoint.scale,
                                   currentJoint.scaleOrientation,
                                   currentJoint.center);
            // Update current matrix with matrix from this joint
            currentMatrix = newMatrix.multRight(currentMatrix);
            // Transform all vertices associated with this joint
            var iNumAffectedVertices = currentJoint.skinCoordIndex.length;
            var v;
            for (v=0; v<iNumAffectedVertices; v++) {
              var newVertex =
currentMatrix.multVecMatrix(coordList[currentJoint.skinCoordIndex[v]]).multiply(currentJoint.skinCoord
Weight[v]);
              skinCoord.point[currentJoint.skinCoordIndex[v]] =
skinCoord.point[currentJoint.skinCoordIndex[v]].add(newVertex);
            }
            // Transform all children
            var children = currentJoint.children;
            var c;
            for (c=0; c<children.length; c++) {
              nextjoint = children[c];
              currentMatrix.getTransform(translation,rotation,scale);
              Transform();
            }
          }
        }
```

Figure 3.2: Transforming vertices related to a joint

Figure 3.3: Screenshot 1 from animated soccer player with advanced kicking gesture

Figure 3.4: Screenshot 2 from animated soccer player with advanced kicking gesture

Figure 3.5: Screenshot 3 from animated soccer player with advanced kicking gesture

Figure 3.6: Screenshot 1 from animated soccer player with advanced kicking gesture

4 Conclusion

The WP 6 T1 has implemented a multi-view compliant player with both photorealistic as well as computer graphics 3D rendering capabilities according to the needs of the project. The use of standardized 3D file-formats for 3D content of avatars is future prove and goes hand in hand with new emerging technologies in web based 3D graphics rendering.

5 Installation

Please unzip the file “gesture_recognition_player_scene” in the Fine Plone server and copy the data to your hard disc. Please install the BS Contact 3D engine, register it for x3d binary files and double click on the file “player_on_ground_X3DB” in the directory “player_scene” to start the animated 3d scene or use the file menu to load. You can change your viewpoint by holding the left mouse button down and move the mouse around the kicking avatar object.

5 References

[1] H-ANIME Standard:
Humanoid Animation Working Group of Web3D consortium
http://h-anim.org/

[2] X3D and HTML5
http://www.web3d.org/x3d/wiki/index.php/X3D_and_HTML5