QUALISYS WEB TRACKER – A WEB-BASED VISUALIZATION TOOL FOR REAL-TIME DATA OF AN OPTICAL TRACKING SYSTEM

Andraž Krašček and Jaka Sodnik
Faculty of Electrical Engineering, University of Ljubljana, Slovenia

Abstract – In this paper, we describe a web-based tool for the visualization and analysis of real-time output data of a professional optical tracking system – Qualisys. Optical tracking systems are used for capturing the positions and movements of various objects in space with the aid of several high-speed cameras and reflective markers. The positions of individual markers are represented as time-dependent 3D spatial points. Tracking software running on a single dedicated computer enables the visualization of data as an interactive 3D scene. The main goal of our work is to enable a real-time visualization of this 3D scene on multiple computers simultaneously with the aid of modern web technologies and tools. The framework is based on an interactive Node.js web server, which streams data from the Qualisys tracking software, reformats it and sends it to a web browser through a fast WebSocket protocol. The web browser enables the visualization of the 3D scene based on WebGL technology. The tool described in this paper enables a synchronized visualization of the tracking data on multiple computers simultaneously and thus represents an excellent teaching and presentation tool of optical motion capture techniques.

1. INTRODUCTION

The term "motion capture" refers to a technique for capturing and recording movements of various objects in space [1][2]. It is widely used in the field of biomechanics, game industry, movie and television production, etc. The majority of widely used motion capture techniques is based on optical methods that record the object's movements. A number of cameras are used to observe the passive or active reflective markers, which have to be attached to the points of interest or to the points that are then tracked in space. The system calculates the exact location of each marker in space by triangulation based on the projection of the marker onto each camera's 2D image plane. Figure 1 demonstrates the basic principle of tracking 3D data based on multiple 2D images.

Several cameras are typically used in order to provide constant visibility of each marker by a minimum of three independent cameras. The high number of cameras provides redundancy, a lower possibility for marker loss and occlusion, and also a higher accuracy over the entire tracking volume.

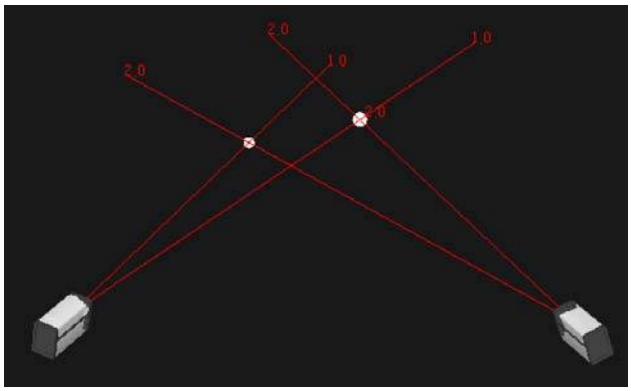

Figure 1. The basic principle of triangulation – reconstruction of 3D data from 2D images [3].

The passive markers are made from simple, light and highly-reflective plastic materials and require an active source of infrared light within each camera. The infrared light is then reflected from the markers back to the cameras and provides good contrast between the markers and the surroundings. Active markers, on the other hand, can be simple LEDs, which do not require an additional source of light within each camera.

In our research, we deal with a professional motion capture system Qualisys [4], which consists of eight high-speed cameras, a set of passive markers and a dedicated tracking software called Qualisys Track Manager (QTM) [5]. The latter is responsible for calculating the exact 3D location of each marker from 2D images acquired by individual cameras. It is a desktop application that runs on a normal PC and communicates with the cameras through standardized Ethernet protocol. It is operated through a well-designed and intuitive GUI and enables a real-time visualization of all tracked points in a virtual coordinate system. It also enables the control over all cameras, such as calibration, timing, capture rate, exposure and flash time, etc.

Each marker (3D point) is presented with three independent coordinates (X, Y and Z) in a Cartesian coordinate system. It is illustrated as a colored dot at the corresponding spatial position. A set of markers can be grouped into a "model". The model is based on a certain number of markers, attached to different parts of the object tracked. The rigid parts of the object (the parts with a constant inter-distance) within the model can be presented with "bones". The bones can be built with the aid of GUI by selecting individual markers in the model

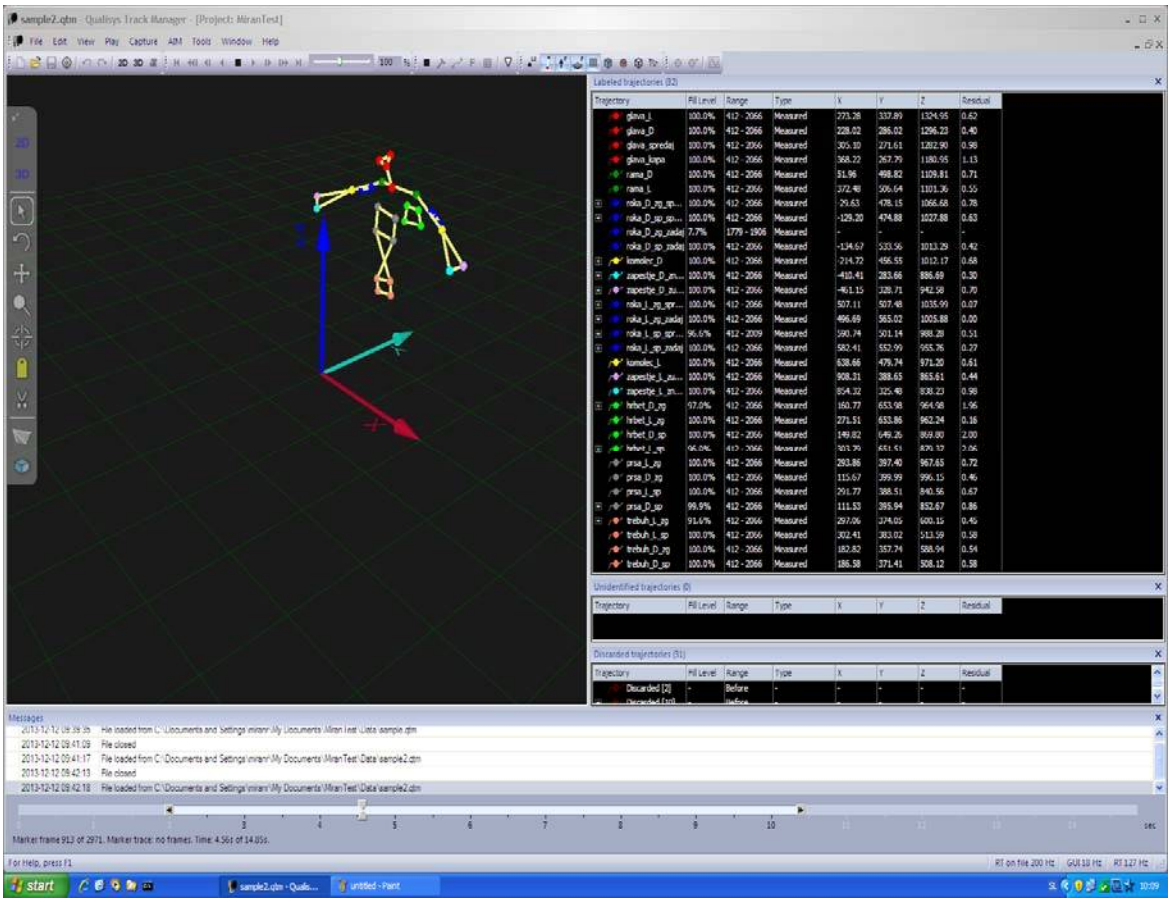

Figure 2. The visualization of an AIM model in QTM software consisting of a set of markers attached to different parts of a human body.

and connecting them with straight lines. Figure 2 shows an example of a set of markers attached to a human body. The final model with a selected number of markers and bones can be saved as an "AIM model" (Automatic Identification of Markers), which is then applied in future measurements. This process helps the QTM software to increase the tracking accuracy with the aid of exact distances and relations between the markers.

The measurements of marker locations and model movements can be saved and processed locally or streamed to an additional computer through a predefined RT (Real-Time) protocol [6]. The "RT protocol" feature enables the QTM software to function as a TCP or UDP server from which various clients can acquire tracking data in real time.

The visualization and analysis of tracked objects and models can currently be performed only on the computer running the QTM software. In this paper, we propose a complete solution for the visualization and analysis of the tracked data on multiple computers simultaneously, in real time and even from a remote location. It is based on modern web technologies and protocols, which enable full duplex client-server communication and real time rendering of 3D models and objects in a browser. In the following sections we provide an overview of the key web technologies used for this experiment and describe all major components of the system and communication protocols between them.

2. QTM REAL-TIME SERVER PROTOCOL

RT protocol component enables the retrieval of processed real-time data from QTM over a TCP/IP or UDP/IP connection [6]. The protocol structure is well defined and provides all the basic features such as: auto discover, settings changing, streaming, error messaging, etc. A client that connects to the server can require data in 2D, 3D or 6DOF (six degrees of freedom), with or without the marker labels, etc. In our case, we use the server to stream 3D points of a set of predefined markers in real-time.

3. WEB SOCKET

The web was originally designed around a request/response model of a HTTP protocol, which then evolved to asynchronous XHR request (part of AJAX) making the web more dynamic. However, the updates from the server are still only received upon client request triggered by user interaction or periodic polling. This carries big data overhead and higher latencies. WebSocket API is a part of HTML5 specification and introduces a socket-like persistent full-duplex connection between the

server and the client. The data can be sent from the server to the client and vice versa without prior request. After the initial handshake, the data is sent in frames with a frame header in the size from 48 bits to a maximum of 112 bits, depending on the payload data size. The maximum frame size depends on the implementation of the protocol. [7]

3. WEBGL

WebGL is a royalty-free, cross-platform API that brings OpenGL ES 2.0 to the web. It is implemented as a 3D drawing context within HTML, exposed as a low-level Document Object Model interface. It uses the OpenGL shading language, GLSL ES, and can be cleanly combined with other web content layered on top or underneath the 3D content. It is ideally suited for dynamic 3D web applications in the JavaScript programming language, and has been fully integrated in all leading web browsers. [8]

Rendering one frame in WebGL can be computationally expensive and can block the JavaScript event loop; therefore rendering should not be implemented by using the SetInterval or infinite loop but rather with the requestAnimationFrame method. This method will execute rendering script at the first available time slot without blocking the user interface or other scripts.

4. NODE.JS

Node.JS was first introduced in 2009 by Ryan Dahl at a JavaScript conference in Berlin. It is a JavaScript based web server. Thanks to Google's fast JavaScript interpreter and a virtual machine called V8, used by Google in its Chrome web browser, Node.JS can outperform traditional server stacks. With classic scripting programming languages, such as for example PHP or ASP, the programmer writes scripts that are executed by a server installed separately. In the case of Node.JS, the programmer writes a code that represents a part of the server itself.

Node.JS executes the code in same process on a single thread called event loop. One event is performed in each loop, so all the I/O events must be written in an asynchronously non-blocking way. A good example of such a code is a database query. Node.JS sends a query to a database that takes a considerable amount of time to process. In the blocking way, the server would wait for the database response and only then continue with the execution. On the other hand, the non-blocking script would not wait for the database response, but would handle other events instead. The response is pushed in the event queue immediately when it is returned by the database. In this way, Node.JS is capable of handling a high number of simultaneous connections.

The core of Node.JS is just a set of low-level APIs, known from JavaScript and V8 JavaScript implementation. Additional functionalities can be added as modules via NPM (Node Package Manager).

5. SYSTEM ARCHITECTURE OF QUALISYS WEB TRACKER

Our system consists of three main entities: QTM software, Node.JS server and multiple clients with ordinary browsers. The coordinates of all the tracked markers are transferred from QTM to Node.JS server over the RT protocol. The server handles the data translation from raw stream to JavaScript Object Notation (JSON) and broadcasts it to all connected clients. Each individual client is responsible for the reconstruction and visualization of the 3D scene from the received data. The following two subparagraphs briefly describe the basic functionalities of the server and the client.

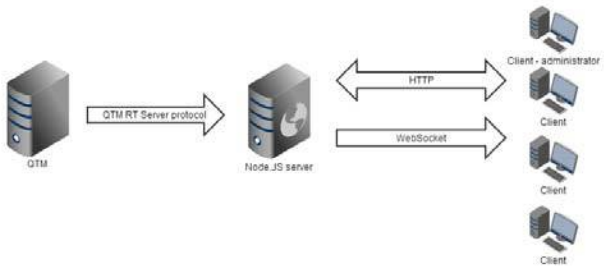

Figure 3. The basic architecture of the Qualisys Web Tracker software.

Server side

The application's server side consists of three main modules: HTTP module, WebSocket module and RT protocol module. The HTTP module is a basic Node.JS module which handles HTTP requests. In our application, it is mainly used for handling the delivery of static files (like CSS, JS, etc.) and control parameters in the administrator view. The other two modules are used to transfer the data from QTM to the clients through the server. The RT protocol module handles the communication between the QTM and the server application, while the WebSocket module communicates with multiple clients simultaneously.

Before broadcasting any data to potential clients, an administrator has to connect to the QTM through a special administrator view. The administrator view is a simple web application running on a separate port and requiring user authentication. The administrator specifies the corresponding QTM IP address and port. After connecting, the administrator requests a special XML file which specifies all the important properties of the markers, such as for example their labels (names) and colors. The specification of the properties is important in order for the markers to be correctly interpreted and visualized by the clients. It is vital that this XML file is transmitted before the start of the data stream, so that it can be parsed and saved in memory, otherwise the data with the marker coordinates sent to the client will be

inadequate. The administrator can then control the stream (start and stop) and also set the transmission rate (in samples per second). The default value for the transmission rate is set to 60. The initiation or stopping of the stream by the administrator actually starts and stops the data stream between the QTM and the server. The main functionality of the server application is to translate the coordinates data from raw buffer stream to JSON format, a typical format for representing data in JavaScript. One 3D frame becomes an array with the size corresponding to the number of nodes. Each item is an object with x, y, z coordinates expressed in millimeters and the residual keys. Some additional parameters are added to JSON format, such as frame number and type, which may be used in further development. The data array is then broadcasted to all connected clients via WebSocket. Figure 4 represents an example of a frame in JSON format, which is broadcasted through WebSocket protocol.

```
...,
[
    components: [
        0: {
            count: 32,
            dr: 0,
            markers: [
                0: {
                    residual: 1051826270,
                    x: 354.6088562011719,
                    y: 331.4305725097656,
                    z: 1318.083740234375
                },
                ...
            ],
            name: "3D Component",
            oosr: 0,
            type: 9
        }
    ],
    frameNo: 846,
    timestamp: 2.0874274e-317
],
...,
```

Figure 4. The representation of marker data in JSON format

Client side

The client side application consists of two parts, a WebSocket and WebGL module. The WebSocket module is very simple. After loading the application and all its external dependencies (JavaScript libraries), the module connects to the Node.JS server. The connection is open for the lifetime of the application. When the data in JSON format is received from the server, it is saved to a local variable. Our goal was to keep the WebSocket module as lean as possible so it does not block the WebGL module.

The WebGL module is responsible for constantly rendering a 3D scene and exposing it in canvas HTML5 element. An infinitive rendering is necessary because of the constant updates of the scene. The updates can come from two events. The first update comes from the WebSocket module, which updates the local variable with new coordinates. This change is easy to detect; however, it is much simpler to continually render the scene because of the second type of event. The latter is the user interaction. The user can jaw, pitch, or zoom in or out the scene. This interaction can also be detected and the scene can be rendered on demand, but we get a much more smooth interaction if we use the requestAnimationFrame method and render the scene whenever possible. We try to achieve a refreshing frequency of 60 frames per second. When the user interacts with the scene, his or her changes are saved in a special view matrix. The view matrix is a transformation matrix that transforms the vertices from scene-space to view-space. The WebGL module is responsible also for applying the vertex and fragment shaders.

6. CONCLUSION

The tool described in this paper enables a remote and interactive visualization of the tracking objects from the Qualisys optical tracking system. The 3d tracking scene can be presented on multiple computers simultaneously and synchronized with the QTM software. It is based solely on web technologies and runs in all modern browsers supporting WebGL API. The current version of the tool enables the visualization of all labeled markers in the scene as well as a list of their current coordinates. The colors of the individual markers can also be applied in accordance with their original color in the QTM software.

Optical motion capture systems are often used as teaching tools in lab exercises with a high number of students. With the aid of our tool, a 3D tracking scene can be streamed to multiple computers simultaneously, while also enabling custom interaction, field of view and zoom for each client. In this way, each student can observe the experiment more actively and customize his or her own perception. The tool could easily be integrated in an e-learning framework enabling remote participation in such experiments in real-time.

Currently, no information about the existing AIM models is streamed through the RT protocol component and the visualization of the bones between the markers has to be done manually (hardcoded) on the client. Our main goal for future development is to enable the capability of rebuilding the AIM models or building new ones on the client. The user should be able to select individual markers and connect them with bones. These new models should then be saved on the server or locally on the client itself for future use.

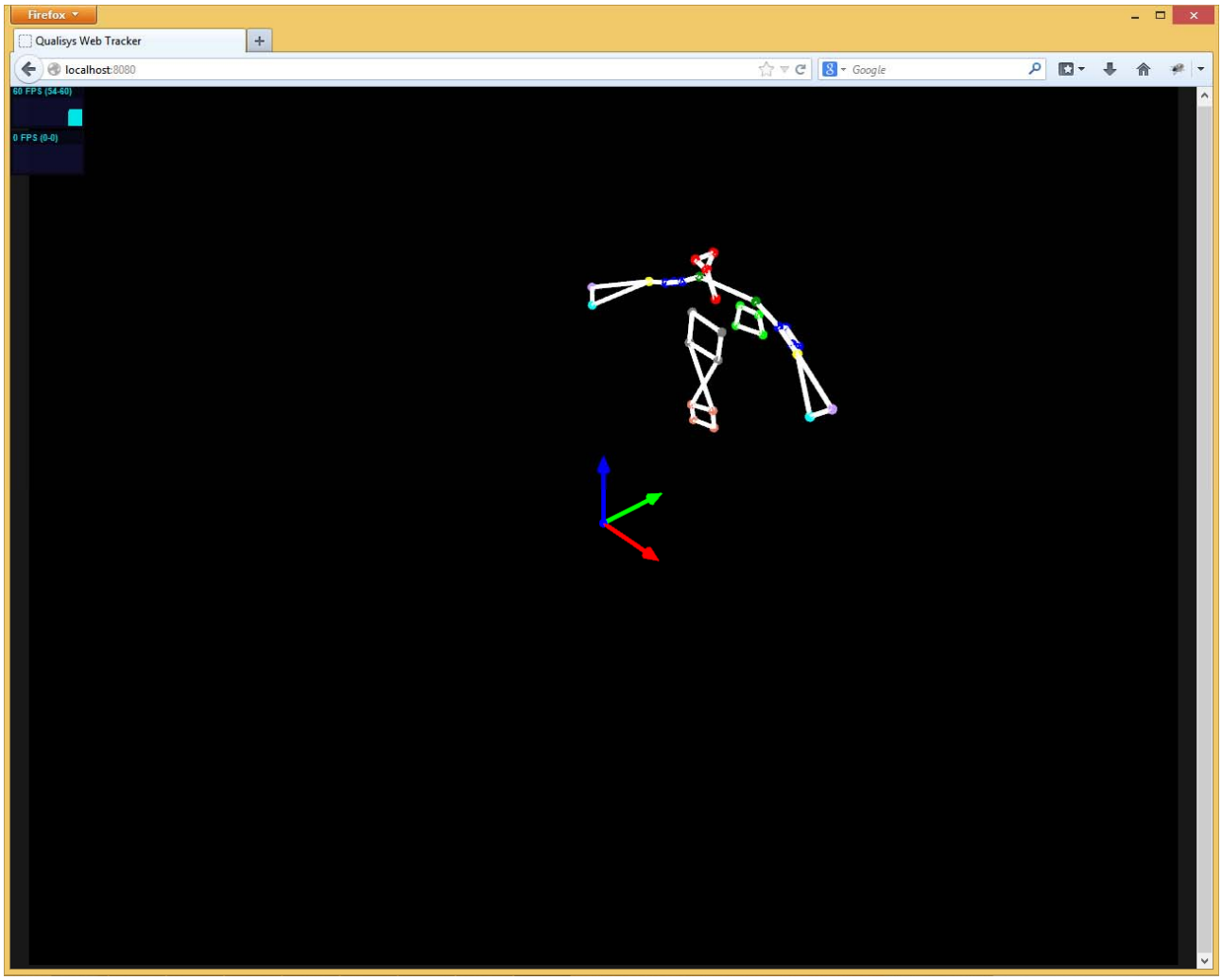

Figure 5. The visualization of an AIM model in a Firefox browser (the same model is shown on figures 2 and 5)

REFERENCES

[1] A. G. Kirk, J. F. O'Brien, D. A. Forsyth, "Skeletal parameter estimation from optical motion capture data," *IEEE CVRP 2005*, vol. 2, pp. 782-788, 2005.
[2] M. Gleicher, N. Ferrier, "Evaluating video-based motion capture," *Proceedings of Computer Animation 2002*, pp. 75-80, 2002.
[3] S. Hofverberg, "Theories for 3D reconstruction," *Qualisys QTECH1004 v1.0*, 2007.
[4] Qualisys – Motion Capture Systems, http://www.qualisys.com/, 12/2013.
[5] QTM – Qulisys Track Manager, *User Manual*, 2011.
[6] L. Nilsson, "QTM Real-time Server Protocol Documentation, V1.9," 2011.
[7] The WebSocket API, http://www.w3.org/TR/websockets/, 12/2013
[8] WebGL - OpenGL ES 2.0 for the Web, http://www.khronos.org/webgl/, 12/2013