Open Communication with Quest3D

- Draft-

1 Introduction

Quest3D ships with some communication support, including Network, Serial, Socket, User Input, ActiveX, File-Handling (subject to Quest3D edition restrictions).

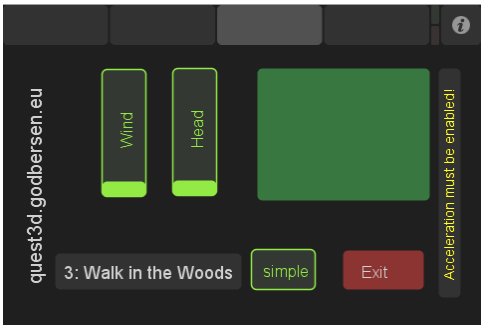
- User Input addresses peripheral devices (keyboard, mouse, and joystick),
- Network refers to a symmetrical DirectX use.
- Serial ports are partially outdated or require USB2serial converter software,
- Socket support is limited to TCP client use,
- ActiveX is platform-dependent,
- File handling is mainly used to support persistence.

This paper discusses some available extensions to allow for richer and more open forms of communication from and to Quest3D.

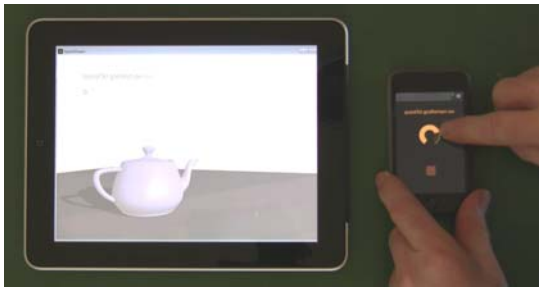
- TCP Server with text support
- UDP support
- OSC support
- TUIO support
- Win7 Touch support

2 The Need for Open Communication Support is Rising

The advent of Smartphones and tablet computing is a typical example. These devices provide inexpensive consumer level sensors of many kinds (accelerometer, orientation, magnetic field), opening a new horizon on inexpensive and ubiquitous HCI interaction. Some Apps like “touchOSC” provide layout editors. This allows customizing a GUI without programming.



Furthermore, the pressure of the Quest3D engine being stuck to DirectX is reduced. Placing all GUI elements on a smartphone/tablet device and streaming the screen output back to such a device makes DirectX invisible (see my video examples).



Open forms of local interprocess communication enable ad-hoc experiments and custom solutions. TCP or UDP transport is an easy and open way to achieve this.

3 Background: Serialization

http://en.wikipedia.org/wiki/Serialization In computer science, in the context of data storage and transmission, serialization is the process of converting a data structure or object into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and "resurrected" later in the same or another computer environment. When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object.

To "serialize" an object means to convert its state into a byte stream in such a way that the byte stream can be converted back into a copy of the object.

This process of serializing an object is also called *deflating* or *marshalling* an object. The opposite operation, extracting a data structure from a series of bytes, is *deserialization* (which is also called *inflating* or *unmarshalling*).

Various methods are on the market. Keywords are: Network Byte Order, plain text, CSV, XML, OSC, ActiveX, …

4 Open Access to Quest3D Data Structures

The first candidates are the basic data types accessible: Value, Vector, Matrix, and Text. Furthermore the generic channel “Buffer”, which holds a byte stream. The interpretation is up to the participating channels.

5 Raw Network Transport

A choice between TCP and UDP. Often plain text ASCII is used as a transfer encoding. This reduces transcoding problems across different hardware architectures.

My plug-ins addressing this topic are SocketServerText, UDPreader and UDPwriter. I use them so far mainly for CSV encoded data structures, access to Flash XMPsocket, and as a transmission vehicle for OSC packages.

6 OSC

http://opensoundcontrol.org/spec-1_0 Open Sound Control (OSC) is an open, **transport-independent, message-based protocol** developed for communication among computers, sound synthesizers, and other multimedia devices.

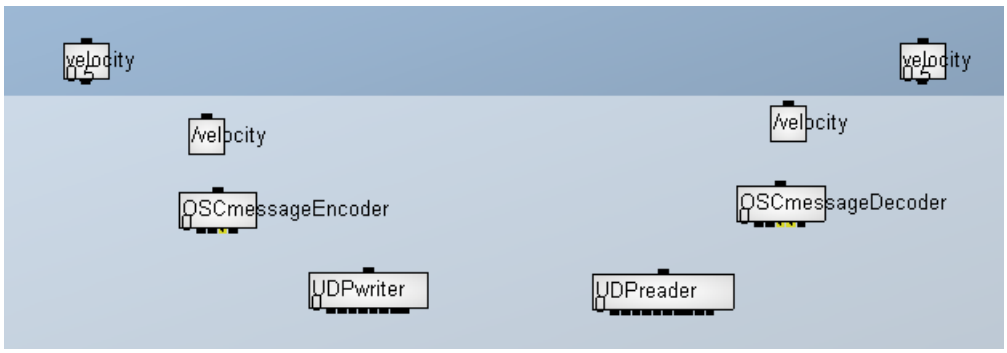
The unit of transmission of OSC is an ***OSC Packet.*** Any application that sends OSC Packets is an ***OSC Client***; any application that receives OSC Packets is an ***OSC Server***[1]. An OSC packet can be naturally represented by a datagram by a network protocol such as UDP. The contents of an **OSC packet** must be either an ***OSC Message*** or an ***OSC Bundle***. An OSC message consists of an ***OSC Address Pattern*** followed by an ***OSC Type Tag String*** followed by zero or more ***OSC Arguments***. An OSC Bundle consists of the OSC-string "#bundle" followed by an *OSC Time Tag*, followed by zero or more ***OSC Bundle Elements***. An OSC Bundle Element consists of its *size* and its ***contents***. The contents are either an OSC Message or an OSC Bundle.

My solution is to provide a set of plug-ins: OSCpackageDecoder is able to decode bundles, OSCmessageEncoder deserializes OSC messages into sets of native Quest3D Value and,Text channel. OSCmessageEncoder allows to serialize the content of a set of Text and Value channels. Packaging of OSC bundles is not jet supported.

Prinicipal scenario:

Imagint you want to send a variable “velocity” with a value of 0.5 from application A to B.

Application A Application B



[1] Confusing to some. I avoid using these terms.

OSC does the following The name of this variable is transported as an Address Pattern "/velocity". The data type of the Quest3D value channel is recognizes as a 32 Bit real. This will be noted as an "f" in the Type Tags list. The value itselv is encoded in "network byte order".

The package send over the UDP network contains the following components:
Address Pattern "/velocity", TypeTag "f" and the attribute "0.5" (binary encoded).

This serialization contains all information to identify and read "velocity" on any computer architeture or operating system.

It's good practise to agree on sending normalized data, e.g. normalized mouse positions.

It's up to the sender to decide, when or how often "velocity" is transmitted. Note that commands are produced in the same fashion: For example, send a "/start" with the attribute 1.

7 TUIO

TUIO stands for Tangible User Interface Objects, and is considered as a community standard. TUIO enabled trackers are available in many varieties, including apps on iOS and Android, and full size touch screens.

http://reactivision.sourceforge.net/#usage **reactIVision** is an open source, cross-platform computer vision framework for the fast and robust tracking of **fiducial markers** attached onto physical objects, as well as for **multi-touch finger tracking.** It was mainly designed as a toolkit for the rapid development of table-based tangible user interfaces (TUI) and multi-touch interactive surfaces. reactIVision sends TUIO messages via UDP port 3333 to any TUIO enabled client application. See http://www.tuio.org/?specification

All ingredients of decoding TUIO are delivered with my OSC channels.

However, the state reconstruction requires an extra effort. My sample application invoke LUA scrips and Quest3D arrays to achieve this.