VRPN: A Device-Independent, Network-Transparent VR Peripheral System

University of North Carolina at Chapel Hill

Russell M. Taylor II
Thomas C. Hudson
Adam Seeger
Hans Weber
Jeffrey Juliano
Aron T. Helser
(and more than a dozen others at UNC and elsewhere)

Brought to You By

- The U.S. National Institutes of Health
  - National Center for Research Resources
    - » UNC Resource on Computer Graphics for Molecular Studies and Microscopy
- Commercial support provided by 3rdTech
  - Walt Disney VR Studios
  - Schumberger Cambridge Research
  - Paid-for improvements are in the public domain

"The great thing about standards…

…is that there are so many to choose from"

- Commercial:
  - vrco's CAVELib, Division's dVS, Sense8's WorldToolKit, Disney's Panda3D, …
- Research:
  - MR toolkit, GIVEN++, DIVE, BrickNet, Alice/DIVER, AVIARY, Maverik/DEVA, VR Juggler, Bamboo, Dragon, DIVERSE, Vlib, …
- Why on Earth publish another one?!?

VRPN is *not* a complete VR API

"You can only standardize what nobody cares about"

- Does not provide user interaction techniques
- Does not provide a scene graph (or transform tree)
- Does not provide graphics techniques

- Does:
  - handle the device control layer
  - provide communication between hosts
  - hook to higher levels of a VR system

VRPN System Vision

"A chicken for every pot and a PC for every VR device"

- Provide Easy Access
  - » Common Interface to different devices
  - » Access to VR peripherals from any graphics engine
  - » Multiple simultaneous connections to a server
  - » Storage and replay of interactive sessions
- Increase Robustness
  - » Keep the (sometimes fragile) trackers working
  - » Client application survives failure/restart of device server
- Be Efficient
  - » Timing: Low latency plus time stamps on common clock
  - » Communications Efficiency

Example VRPN setup

PC Linux

Parallel

Hand Buttons

Parallel

Hand Buttons

Ethernet

RS-232

FOB Sensor

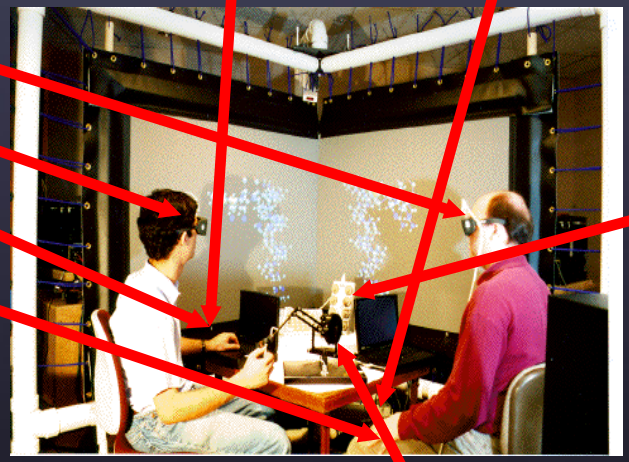
FOB Sensor

FOB Sensor

FOB Sensor

FOB Source

SGI Irix Simulate Render



SGI Dial/ Button

RS-232

SGI Irix

Ethernet

PC Win32

PCI

PHANToM

Ethernet

Demonstration

- Didn’t bring the PIT with me
- Did bring a stored file of tracker values

VRPN Guideline

"Make difficult things easier…"

- Writing client applications for arbitrary devices
- Running servers for multiple devices

"…without making easy things more difficult…"

- Warning/error messages from device driver
- Writing a driver for a new device

"…or making desirable things impossible"

- Special commands for specific device drivers

“Making difficult things easier…”

- Opening a (possibly networked) connection to an arbitrary type of tracking device and getting locally-time-stamped reports from it, restoring the connection if the server is shut down and restarted

```
#include “vrpn_Tracker.h”

void handle_pos(void *, const vrpn_TRACKERCB t) {
  printf("Position of sensor %d is %f, %f, %f\n",
         t.sensor, t.pos[0], t.pos[1], t.pos[2]);
}

main() {
  vrpn_Tracker_Remote *tkr = new vrpn_Tracker_Remote(“Tracker0@myhost”);
  tkr->register_change_handler(NULL, handle_pos);
  while (1) { tkr->mainloop(); }
}
```

“Making difficult things easier…”

- Starting multiple tracker, button, and analog servers sharing a common network link to each of multiple applications, logging their sessions on a per-connection basis as requested by the clients

```
cat > vrpn.cfg
  vrpn_Tracker_Flock   Tracker0       4       /dev/ttyC1     115200
  vrpn_Tracker_Fastrak Fastrak0               /dev/ttyC2     19200
  vrpn_Tracker_Fastrak Isense900              /dev/ttyC3     115200 /
    Wand               Wand0   0  -1.0 0.0 0.0 1.0  -1.0 0.0 0.0 1.0 /
    Stylus             Stylus0 2
  vrpn_Magellan        Magellan0              /dev/ttyC4     9600
^Z

vrpn_server –f vrpn.cfg
```

"…without making easy things more difficult…"

- Dealing with remote devices
  - This was not included at first, and it *really hurt*
  - Determining if device open was successful
  - Error and warning messages from device
- Local client and server
- Writing a driver for a new device
  - vrpn_BaseClass
  - Root classes for functions, and subclasses
    - » Serial tracker class
    - » Analog server class

"…or making desirable things impossible"

- A very real danger in implementing libraries
  - One ditch: Don't want the "lowest common denominator" device
  - The other ditch: Don't want a combinatorial explosion of special-case code to deal with different devices
  - Ditch avoidance described next…

Key Idea: Device Factoring

"Don't think of writing drivers for a set of devices – think of designing interfaces for a set of functions

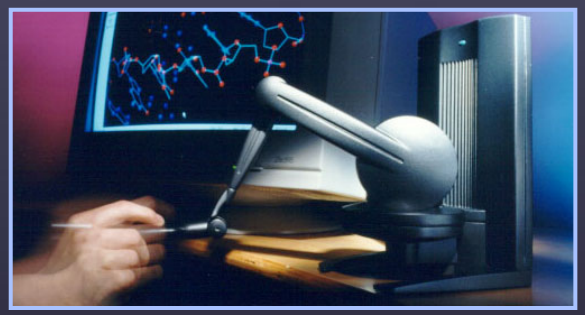
- Each device is factored into one or more interfaces
  - Client connects to the different interfaces separately
  - VRPN maps multiple data streams through the same connection when appropriate, based on run-time bindings
- Factoring and extensibility are supported by:
  - Devices silently ignoring unrecognized message types
  - Layered devices
  - Multiple-behavior devices
  - Application-level access to all messages

Factoring: Common Types

- "Input" devices
  - Tracker: 1 or more sensors, pos/quat for each
  - Button: 1 or more buttons, press/release for each
  - Analog: 1 or more floating-point *values*
  - Dial: 1 or more floating-point *incremental angles*
- "Output" devices
  - Sound: Audio clips with volume, location, etc
  - ForceDevice: Surfaces and forces in 3D

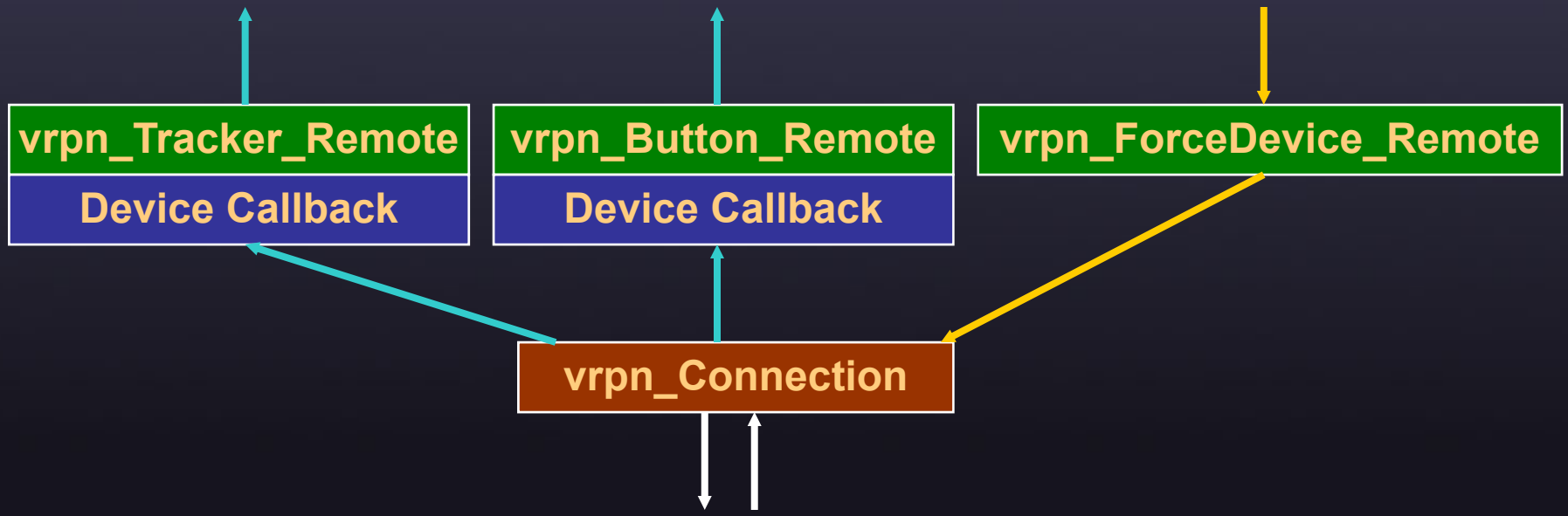
Factoring Complex Devices

- SGI Dial/Button Box is 32-in Button, 8-in Analog
- Phantom consists of a single-sensor Tracker, a single Button, and a ForceDevice
- Intersense consists of multiple-sensor Trackers, each wand is also a 2-in Analog and 5-in Button, each stylus is also a 5-in Button
- UNC Joystick consists of 7-in Analog and 2-in Button



Client-side VRPN

- App connects to *Remote* objects for each device
- VRPN consolidates devices on connections
- Raw message callbacks marshaled by "input" objects into proper callback types for each device
- Methods on "output" devices pack messages to server object

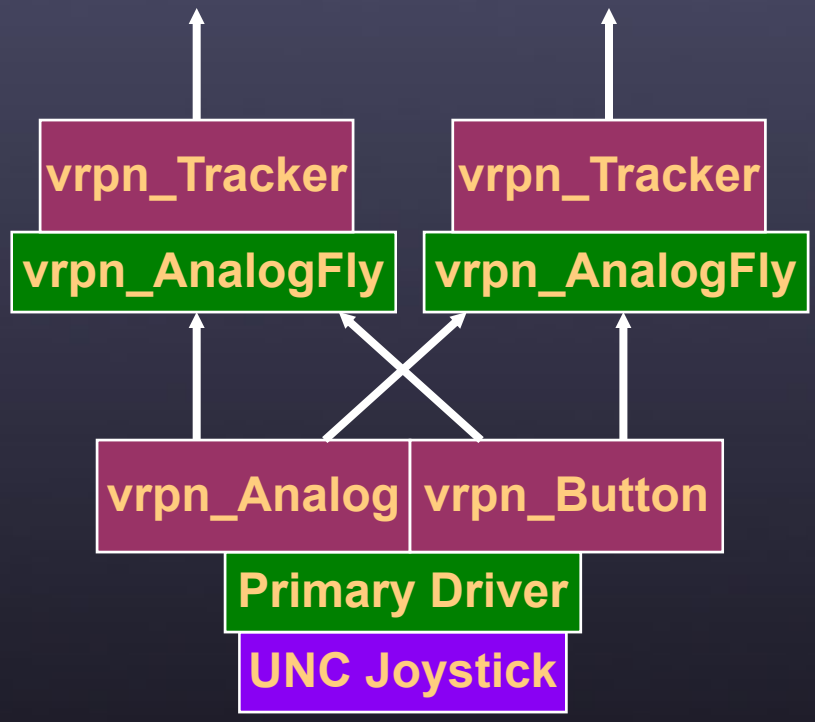


User's view of VRPN: Phantom Example

```
#include "vrpn_Tracker.h"
#include "vrpn_Button.h"
#include "vrpn_ForceDevice.h"
… callback functions here
main() {
  vrpn_Tracker_Remote *tkr =
       new vrpn_Tracker_Remote("Phantom@myhost");
  vrpn_Button_Remote *btn =
       new vrpn_Button_Remote("Phantom@myhost");
  vrpn_ForceDevice_Remote *frc =
       new vrpn_ForceDevice_Remote("Phantom@myhost");
  … register callback handlers here
  while (1)
     {tkr->mainloop(); btn->mainloop(); frc->mainloop();}
}
```

Exporting Multiple Interfaces: The UNC Joystick Box

- Layered Device
  - Exports Analog/Button and Tracker
- Multiple-behavior Device
  - Exports two Trackers with different behaviors
- Client connects to one or more devices of interest
- Links can be within the same server process, or network links from client to server



Other VRPN Features

- Reliable vs. low-latency class of service
- Synchronized clocks between endpoints
- Logging of sessions (client and/or server)
- Multiple connections to a server
- Client and server reconnect
- Connection accept/close messages
  - Local messages from the vrpn_Connection
- Can run client and server on different machines, as different processes on the same machine, or within the same process

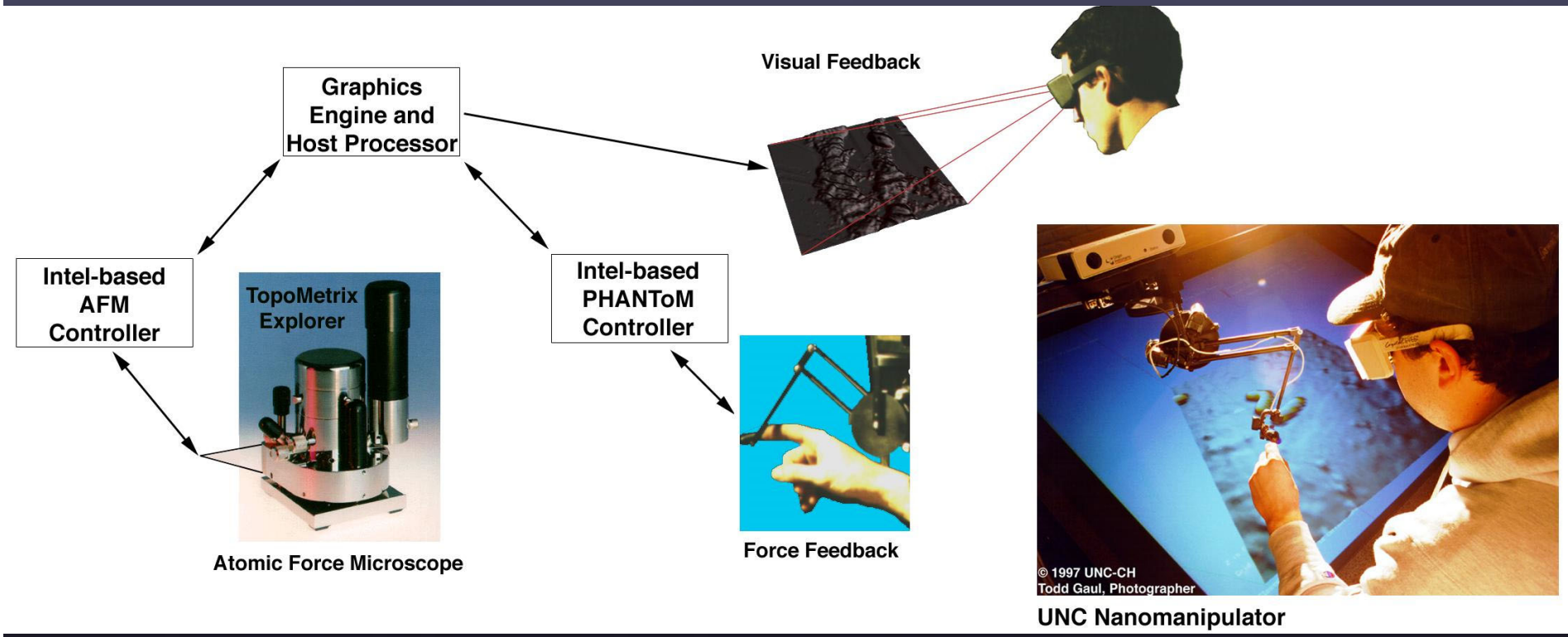
Separate Client and Server When…

- Update loops have very different rates
- Server initialization takes a long time
- Knowing the time of an event is critical
- Server requires frequent access to its device

Performance

- Overhead due to network
  - 0.5ms network, 1.7-3.3ms app-to-app 1-way average
- Reductions compared to “out-of-the-box” drivers
  - Serial port accelerations (buffer, jiffies, Cyclades) together reduce average latency by about 11ms
  - Optimized, time-aware drivers (first-character timestamp, no pessimistic polling)
  - Much faster initialization when connecting to a remote server
- Can get a net gain (*reduction* in latency) using VRPN
  - even over a network connection
- Can get more accurate timestamps

Example Application



Where do I Learn More?

- Web page at www.cs.unc.edu/research/vrpn
- Can download source code
  - Clients run on PC/Win32, SGI/Irix, PC/Linux, Sparc/Solaris, HP700/Hpux, and PowerPC/AIX
  - *Trackers:* Ascension Flock of birds (single or multiple serial lines), Polhemus Fastrak, Intersense IS-600 and IS-900 (including wands and styli), Origin Systems DynaSight, Phantom™, 3rdTech HiBall 3000, Logitech Magellan, and Radamec SPI (video/movie camera tracker).
  - *Other devices:* Logitech Magellan (analog values and buttons), B&G systems CerealBox (buttons, dials, sliders), NRL ImmersionBox serial driver (buttons), Wanda (analog, buttons), National Instruments A/D cards, Win32 sound server based on the Miles SDK, SGI button and dial boxes, the "Totally Neat Gadget" (TNG3) from Mindtel, and the UNC hand-held Python controller (buttons).
- VRPN is in the public domain – use it however you like