Quelloffene Lösungen für die Erweiterte Realität
Verfahren und Implementierungen

Lubosz Sarnecki, Collabora Ltd.

Obwohl das Feld der Virtuellen und Augmentierten Realität seit den 90er Jahren besteht, erleben wir in den letzten Jahren eine rasante marktgetriebene Entwicklung in diesem interdisziplinären Bereich. Zwar ist es historisch von proprietärer Software geprägt, genießt aber eine enthusiastische Community von Open Source Hackern und Unternehmen, die freie Treiber und Middleware entwickeln. Offene Standardisierungs-Bemühungen werden derzeit von der Khronos-Gruppe durchgeführt.

Begriffserklärung XR

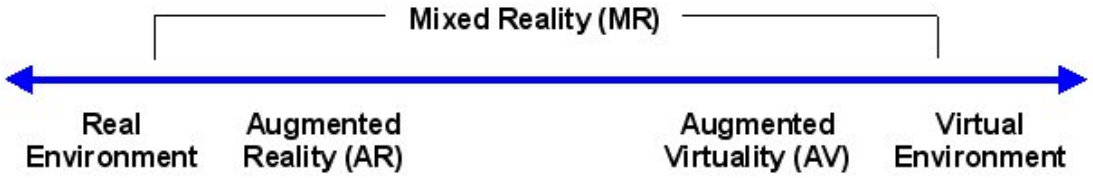

Abb 1: Das Reality-Virtuality Continuum von Milgram, 1994

Der Begriff XR (Extended Reality) wird gerne verwendet um die Aspekte von virtueller und augmentierter Realität zu vereinigen. Er ist hierbei als Synonym von Mixed Reality zu sehen, welches den mittleren Teil der Achse des *Reality-Virtualty Kontinuums* von Milgram einnimmt. Das Kontinuum beginnt bei der vollständigen echten Realität, geht über die augmentierte Realität, bei der virtuelle Gegenstände in die Realität eingefügt werden, und endet in der vollständigen Virtualität.

Tracking Technologien

Neben des Renderings ist das Tracking der Hauptbestandteil von XR Systemen. Im Folgenden werden einige in Konsumentensystemen gängige Verfahren vorgestellt.

Inertiale Messeinheit

Die IMU (engl. Inertial Measurement Unit) ist eine Kombination mehrerer Trägheitssensoren und wird durch Ihre Preisgünstigkeit als Fundament der meisten Trackingsysteme genutzt. Die wichtigsten Sensoren hierbei sind das Akzelerometer (Beschleunigungssensor) und das Gyroskop (Drehratensensor). Viele IMUs enthalten auch ein zusätzliches Magnetometer (Magnetfeldsensor), welches jedoch aufgrund seiner Störungsanfälligkeit auf elektromagnetische Strahlung oft in Implementierungen vernachlässigt oder gar nicht erst verbaut wird. Durch den Einsatz von IMUs in den meisten Mobiltelefonen können Projekte wie Google Cardboard das Smartphone als VR Komplettlösung nutzen und müssen lediglich einen Papierkarton mit Plastiklinsen als Halterung anbieten.
Um aus einer IMU eine vollständige Orientierung zu erhalten muss man mehrere Sensoren zusammenführen.
Eine Implementierung für das Auslesen der IMU Signals der meisten gängigen Konsumenten-VR-Headsets ist *OpenHMD*[4]. Die USB Geräte werden hierbei mit

der Bibliothek *hidapi* angesprochen. OpenHMD verfügt ebenfalls über eine Sensor-Fusion und liefert über eine C-Schnittstelle eine fertige Pose des Kopfes. OpenHMD ist unter der Boost Lizenz veröffentlicht und verfügt über eine aktive Entwicklergemeinde.
Obwohl die IMU prinzipiell für das Bestimmen einer Position im 3D-Raum verwendet werden kann, liefern gängige Sensoren Daten, die zu verrauscht sind und zu sehr driften, um für diesen Zweck zu genügen. Üblicherweise wird mit IMUs deshalb nur eine Rotation bestimmt, während die folgenden Trackingsysteme auch die Position liefern.

Externes Bildbasiertes Tracking
Eine gängige Methode um eine 6DOF (6 Degrees Of Freedom) Pose zu bestimmen, also eine Pose mit 6 Freiheitsgraden, ist das externe bildbasierte Tracking. Hierfür gibt es zwei Ansätze.

Beim ersten Ansatz beinhaltet das Headset die Aktoren und der externe Tracker ist eine Kamera. Beispiele für diesen Ansatz sind Oculus DK2 / CV1, PSVR und OSVR HDK. Bei diesen Geräten findet man ein LED Gitter am HMD und an den Controllern, welches im infraroten Spektrum leuchtet. Durch Blob-Detektion lassen sich die Aktoren, deren 3D Anordnung auf dem Headset bekannt ist, im 2D Kamerabild bestimmen. Mit Algorithmen wie PnP (Point-N-Point), was beispielsweise in *OpenCV* implementiert ist, kann von den 2D Positionen im Kamerabild und ihrer 3D Anordnung auf die Pose des Headsets im Raum geschlossen werden. Eine Open Source Implementierung des kamerabasierten externen Trackings findet man in *OSVR*[12] und dem GPLv2 lizensierten *ouvrt*[1].

Beim anderen Ansatz beinhaltet das Headset die Sensoren und ein externer Tracker die Aktoren. Dies kommt beispielsweise beim Lighthouse Tracking System der HTC Vive und anderen Headsets wie StarVR zum Einsatz.
Hierbei emittieren die Tracker durch 2 an Rotoren angebrachten Lasern, ähnlich wie bei Laser-Scannern, ein Signal, welches durch einfache Lichtsensoren am Headset und den Controllern aufgenommen wird. Durch die Fabrikkalibrierte Anordnung der Sensoren die vom Headset ausgelesen werden kann, sowie der Information, zu welchem exakten Zeitpunkt jeder Sensor von einem Laserstrahl getroffen wurde, lässt sich hierbei eine Pose rekonstruieren. Durch Impulse eines LED Arrays und Fotosensoren können sich mehrere Base Stationen synchronisieren, ohne mit der Tracking Software zu interagieren. Das USB Protokoll der HTC Vive wurde vom Project *LighthouseRedox*[2] dokumentiert. Ein Ansatz für eine Open Source Implementierung des Lighthouse Trackings findet man in meinem experimentellen *vive-libre*[13] Treiber, so wie in dem darauf basierenden MIT lizenzierten *libsurvive*[3].

Bei beiden Ansätzen ist eine Steigerung in der Robustheit des Systems mit einer Erhöhung der Trackeranzahl erreichbar.

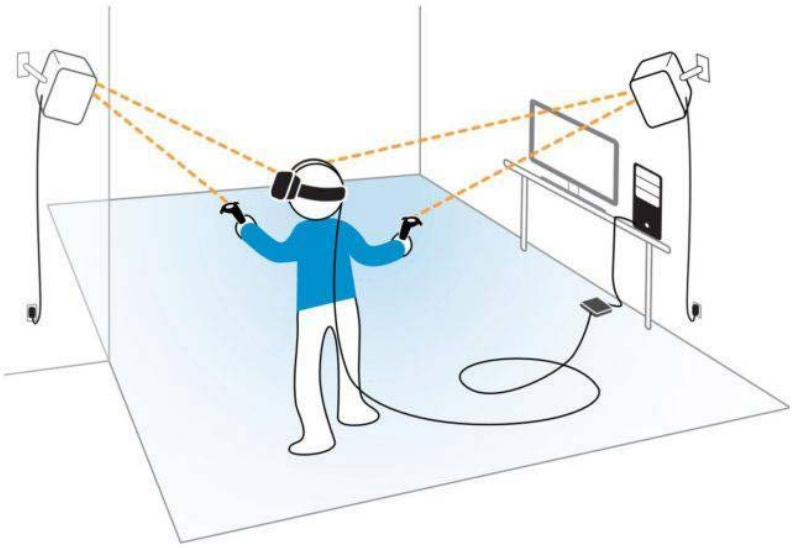

Abb 2: Lightouse Tracking System

Inside Out Tracking
Durch die fehlende Notwendigkeit von externen Tracker-Geräten wird das Inside-Out-Tracking gerne in modernen Konsumenten-HMDs eingesetzt. Hierbei benötigt das Headset lediglich mindestens eine Kamera, die mit Hilfe von aus der Robotik stammenden Verfahren wie SLAM (Simultaneous Localization and Mapping), anhand von Feature-Detektion und Aufbau einer Karte die Pose der Kamera im Raum bestimmen kann. In modernen proprietären AR Implementierungen wie ARcore von Google und ARkit von Apple ist es die bevorzugte Tracking-Methode. Das IMU Signal des Smartphones oder HMDs trägt hierbei, durch seine kleine Latenz und hohe Frequenz, ebenfalls zur Vervollständigung der Ortung bei. SLAM Verfahren, die Trägheitssensorik integrieren, nennt man Visual-Inertial, auch VIO. Leistungsstarke Open-Source Implementierungen sind *ORB-SLAM2*[6] (GPLv3) und *Maplab*[5] (Apache 2.0) von der ETH Zürich.

Für das Tracken der Controller kann das SLAM mit externen Kamera Tracking Verfahren kombiniert werden, wie beispielsweise bei den Microsoft Mixed Reality Headsets. Der Controller verliert hierbei das Positionstracking sobald er aus dem Sichtfeld (Field Of View) der Sensoren austritt und fällt auf seine Trägheitssensorik zurück.
Eine Liste an zahlreichen Open Source SLAM Implementierungen kann auf OpenSLAM eingesehen werden[18].

Handtracking
Mit einem simplen Gerät wie der Leap Motion oder uSens FINGO, welches eine weitwinklige Stereo Infrarot Kamera enthält, kann mit Bildverarbeitung so wie Künstlicher Intelligenz ein solides Tracking von Händen erreicht werden. Leider blieb es bei Open Source Treibern wie *OpenLeap*[7] nur bei minimalen Ansätzen für die Implementierung. Tiefenkameras wie der Microsoft Kinect oder Intel Realsense bieten eine weitere Alternative zum Körpertracking. Das *ROS* liefert Implementierungen für Handtracking auf Tiefenkamerabasis[8].

Übersicht von Standardisierungsansätzen und APIs

OpenVR
Eine sehr weit verbreitete API fuer VR ist Valve‘s *OpenVR*[10]. Zwar ist ihre Implementierung SteamVR nicht Open Source, allerdings bietet SteamVR ein Treiber-Interface an, das die Entwicklung oder Einbindung von Open Source Tracking Treibern ermöglicht. Zum Beispiel liefert das *SteamVR-OpenHMD*[9] Projekt eine Anbindung an alle von OpenHMD unterstützten Headsets.

OSVR
Die von Sensics entwickelte Middleware *OSVR* implementiert externes kamerabasiertes Tracking für die HDK und HDK2 Headsets, sowie einen Kalman Filter für die Filterung der Sensordaten und Bestimmung der Pose. Als IPC und Fundament von *OSVR-Core*[12] wird hierbei *vrpn*[11] verwendet, welches auch weitere Gerätetreiber liefert. *OSVR* verfügt zusätzlich über Anbindungen zu proprietären Treibern wie SteamVR und Oculus VR.

OpenXR
Die Khronos Gruppe publizierte bekannte Industriestandards für Grafikschnitstellen wie OpenGL und Vulkan. Derzeit wird an der Spezifikation der OpenXR API gearbeitet, die Herstellerunabhängige Applikationen ermöglichen wird.

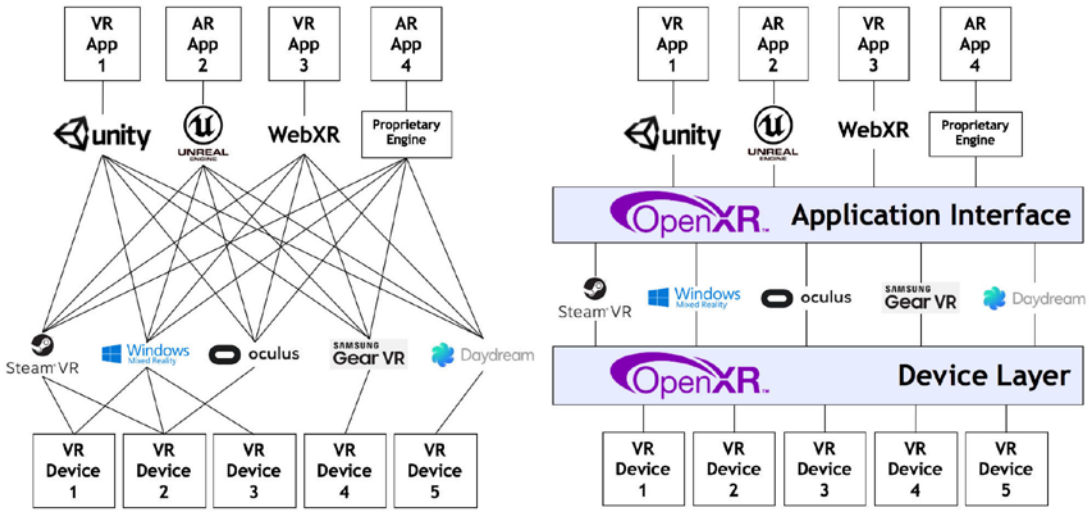

Abb 3: API Fragmentierung vor und nach dem Einsatz von OpenXR

Actions API
Bei klassischen Input Systemen wie SDL 1 wird in der Applikation der Zustand von fest definierten Tasten abgefragt. Dabei kann z.B. die Anwendung geschlossen werden wenn die Escape Taste des Keyboards gedrückt wird oder eine Sprung Aktion in einem Spiel ausgeführt werden wenn die 2. Taste des Spielecontrollers betätigt wurde. Das hat den Nachteil dass eine Neubelegung der Tasten speziell in der Anwendung oder im Treiber implementiert werden muss. Ebenso muss der Applikation zur Implementierungszeit der Controller bekannt sein. Dieses Problem führt oft dazu dass Anwendungen nur einen Controller unterstützen wie z.B. den Microsoft Xbox Controller. Userspace Treiber wie *xboxdrv*[16] helfen hierbei das

Problem zu lösen indem sie für verschiedene Gamepads einen Xbox Controller emulieren. Die Lösung auf Seiten der API ist jedoch einerseits nicht spezielle Tasten abzurufen sondern Aktionen zu definieren, die dann im Treiber einer Taste zugewiesen werden können. Hier werden aus dem oben genannten Beispiel eine Beenden und Springen Aktion die in der Anwendung verarbeitet werden. Der Anwender kann hierbei die Belegung ändern oder auf vorgefertigte Profile zurückgreifen. Da im Bereich der virtuellen Realität eine hohe Inhomogenität und Innovation im Bereich der Eingabe herrscht, ist eine Actions API eine notwendige Lösung. Hierbei können auch Posen des Kopfes, der Controller oder sonstigen getrackten Elemente als Aktionen eingebunden und belegt werden.
Eine proprietäre Implementierung für eine Actions API existiert bereits in SteamVR, welche in OpenVR spezifiziert ist. OpenXR wird ebenfalls eine Actions API besitzen, wie im Vortrag „*Standardizing All the Realities: A Look at OpenXR*“ angekündigt wurde[15].

Direct Mode auf dem Open Source Grafik Stack
XR Geräte profitieren von einem speziellen Anzeigemodus, der Direct Mode genannt wird. Im klassischen Extended Mode behandelt das Betriebssystem das HMD als normalen Monitor – der Desktop wird auf das HMD erweitert, allerdings sind 2D Anwendungen direkt auf dem HMD angezeigt nicht benutzbar. Eine VR Applikation zeichnet hierbei in ein normales Vollbildfenster, das auf dem HMD angezeigt wird. Dies hat auch den Nachteil, dass das Rendering mit relativ hoher Latenz geschieht, da moderne Fenstermanager zum Beispiel Compositing zur Darstellung von Fenstern verwenden, welche nicht auf geringe Latenz ausgelegt sind.

Der Open Source Desktop bietet durch die junge Einführung des DRM Lease Verfahrens von Keith Packard die Möglichkeit Bildschirme komplett vom Desktop Betrieb auszuschließen. Eine Liste mit Bildschirmen, also HMDs, die durch ihre EDID auf „non-desktop“ gesetzt werden, ist im Linux Kernel zu finden. Es sind zahlreiche aktuelle Komponenten wie Kernel, X Server und Mesa Grafiktreiber notwendig, um dieses Feature zu nutzen. Eine genaue Liste an Patches wurde von Christoph Haag zusammengeführt[14].

Eine Möglichkeit DRM Leases zu nutzen ist die Vulkan Grafik API. Durch die Vulkan Erweiterungen `VK_EXT_direct_mode_display` und `VK_EXT_acquire_xlib_display` kann über Xlib Strukte die Bildschirmausgabe im Direct Mode initialisiert werden. Ein Beispiel für die Nutzung des Direct Modes mit Vulkan findet man in meiner VR Grafikdemo *xrgears*[17], welche OpenHMD als Tracker verwendet.

Abkürzungsverzeichnis

VR	Virtual Reality
AR	Augmented Reality
XR	Extended Reality
OpenVR	Valve‘s Open Virtual Reality API
OSVR	Sensics Open Source Virtual Reality Middleware
OVR	Oculus Virtual Reality API
OpenXR	Open Extended Reality Khronos Standard API
SLAM	Simultaneous Localization And Mapping
IMU	Inertial Measurement Unit
OpenCV	Open Computer Vision
6DOF	6 Degrees Of Freedom
ROS	Robotic Operating System
PnP	Point-N-Point
VIO	Visual-Inertial Odomentry
DRM	Direct Rendering Manager
IPC	Inter-process Communication
HMD	Head-Mounted Display

Abbildungsverzeichnis

Literatur- und Quellenverzeichnis

[1] https://github.com/pH5/ouvrt
[2] https://github.com/nairol/LighthouseRedox
[3] https://github.com/cnlohr/libsurvive
[4] https://github.com/OpenHMD/OpenHMD
[5] https://github.com/ethz-asl/maplab
[6] https://github.com/raulmur/ORB_SLAM2
[7] https://github.com/openleap/OpenLeap
[8] http://wiki.ros.org/hand_interaction
[9] https://github.com/ChristophHaag/SteamVR-OpenHMD
[10] https://github.com/ValveSoftware/openvr
[11] https://github.com/vrpn/vrpn
[12] https://github.com/OSVR/OSVR-Core
[13] https://github.com/lubosz/OSVR-Vive-Libre
[14] https://haagch.frickel.club/#!drmlease.md
[15] Nick Whiting, „*Standardizing All the Realities: A Look at OpenXR*“, GDC, March 2018
[16] https://gitlab.com/xboxdrv/xboxdrv
[17] https://gitlab.com/lubosz/xrgears
[18] https://openslam-org.github.io

Autor

Lubosz Sarnecki ist Senior Software Engineer bei Collabora und VR-Enthusiast seit den Oculus DK1 Tagen. Er arbeitet seit 2011 an Open-Source-Software Projekten wie GStreamer. Nach der Arbeit an den GStreamer OpenGL-Plugins konzentrierte sich Lubosz auf Grafik und VR. Er ist Author der GStreamer VR Plugins, zahlreicher Open Source VR Demos und arbeitet derzeit an der Verbesserung des Open Source VR Stacks. Lubosz hat einen B.Sc. in Computervisualistik der Universität Koblenz.

Kontakt

Internet: www.collabora.com
Email: lubosz.sarnecki@collabora.com