TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR SOFTWARE- UND MULTIMEDIATECHNIK
PROFESSUR FÜR COMPUTERGRAPHIK UND VISUALISIERUNG
PROF. DR. STEFAN GUMHOLD

# Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Medieninformatiker

# Open Display Environment Configuration Language

Ronald Großmann
(Geboren am 6. April 1990 in Sebnitz, Mat.-Nr.: 3507432)

Betreuer: Dr. rer. nat. Sebastian Grottel

Dresden, 16. August 2015

## Aufgabenstellung

Die Anzahl der Multi-Display-Installationen nimmt in beinahe jeder Umgebung zu: angefangen von Desktopsystemen mit mehreren Monitoren, über Projektionsflächen mit mehreren Projektoren (aka Powerwalls) bis hin zu komplexen VR-Installationen wie Caves. Oft benötigen solche Display-Systeme leistungsstarke, parallel GPU-Cluster zur Bilderzeugung, allein um der notwendigen Pixelfüllrate gerecht werden zu können. Zusätzlich kommen in solchen Anlagen üblicherweise auch Tracking-Systeme zum Einsatz, die den Benutzer, also die physische Welt, mit den dargestellten Szenen, der künstlichen Welt, verbinden; auch im Desktop-Bereich, z.B. durch Windows Kinect oder Leap Motion. Um solche Systeme zu betreiben bedarf es komplexer Software. Diese ist oft im akademischen Umfeld entwickelt. Solche Software muss umfassend konfiguriert werden, einerseits was die verfügbare Rechnerinfrastruktur betrifft (welche Computer sind mit welchen Ausgabegeräten verbunden, welche Computer dienen rein zur entfernten Bilderzeugung und wie sind die Rechner miteinander vernetzt), andererseits auch was die logischen und physikalischen Parameter der Ausgabegeräte betrifft (virtuelle Desktop-Größen und Teile einzelner Beamer, physikalische Anordnung von Display oder Projektoren und Abgleich mit den Raumkoordinaten eines Benutzer-Trackings). Allerdings hat sich für diese Konfigurationen bisher kein Standard entwickelt.

In dieser Arbeit soll ein Vorschlag für so ein standardisiertes Konfigurationsformat auf Basis von XML entwickelt werden. Mittels XSLT soll es möglich sein, aus einer XML-Datei Konfigurationsdateien für unterschiedliche Programme zu erzeugen. Ein graphischer interaktiver Editor soll das Erstellen und Bearbeiten der XML-Dateien, sowohl der strukturellen Eigenschaften (Computer-Cluster-Architektur, inklusive GPUs und Display-Anschlüssen) als auch der 3D physikalischen Eigenschaften (Display-, Projekt-Setup) anschaulich und einfach ermöglichen. Das XML-Format muss sauber durch Namespaces aufgeteilt und erweiterbar sein, was auch durch entsprechende Funktionen im graphischen Editor reflektiert werden muss (z.B. muss es im Editor möglich sein, eigentlich nicht unterstütze Tags editieren und beim Abspeichern erhalten zu können). Die Erstellung von XSLT-Dateien muss durch den Editor NICHT unterstützt werden, ihre Anwendung zum Export der Konfiguration in entsprechende andere Formate jedoch schon.

Folgende Hardware-Installationen müssen unterstützt werden:

1. Desktop-Computer mit mehreren Monitoren (mindestens zwei) die nicht in einer gemeinsamen Ebene stehen.
2. Stereo-Powerwall durch zwei Beamer betrieben an einem Rechner (Powerwall an der Professur CGV)
3. Großfläche Displaywand mit mehreren Panels (Displaywand an der Professur Multimedia-Technologie)
4. Fünf-Wand-CAVE mit zehn Beamer (im VR-Labor des Lehrstuhls Konstruktionstechnik / CAD)

Hierbei müssen die physikalischen Display- und Projektionsanordnungen unterstützt werden, und zusätzliche Infrastruktur, wie z.B. Computer, GPUs, Tracking-Systeme etc., sollen so weit wie möglich unterstützt werden.

Die Hardwareinstallationen sollen in ihrem physikalischen Raum, in Metern, frei definierbar sein. Ist kein Benutzertracking vorhanden, so muss eine Standardposition für den Benutzer (Blickpunkt) konfigurierbar sein.

Die Konfiguration der Rechnerinfrastruktur muss mindestens die Computer enthalten, die direkt an die Ausgabegeräte angeschlossen sind. Ihre Netzwerkverbindungen untereinander sollten enthalten sein.

Eventuelle Compute-Cluster zur Bilderzeugung und ihre Netzwerkverbindungen untereinander, sowie zu den Ausgaberechnern sollten ebenfalls konfigurierbar sein.

Folgende Software muss unterstützt werden, indem ihre Konfigurationsdateien, mindestens aber der entsprechend dieser Arbeit relevante Teil der Konfigurationsdateien, erzeugt werden kann:

a, MegaMol, bzw. mittels einer kleinen Bibliothek jede an der TUD selbst entwickelte Software

b, Paraview

c, Equalizer (optional)

Weitere Software soll nach Absprache mit dem Betreuer ebenfalls unterstützt werden.

Die Bearbeitung erfolgt mit diesen Teilzielen:

- Anforderungsanalyse auf Basis der vorgegebenen Display-Hardware und Konfigurationsspezifikationen der einzusetzenden Software
- Literaturrecherche zur Large-Display- und VR-Software-Middleware und Konfigurationen. Auch zu allgemeinen Arbeiten zur Kalibrieren und Konfiguration solcher Hardware-System
- Spezifikation der XML-basierten Konfiguration
- Umsetzung des geforderten Editor-Prototyps inklusive XSLT-basiertem Export der Konfigurationen
- Evaluierung im Kontext der vorgegebenen Display-Systeme durch Darlegung und Durchführung des kompletten Konfigurationsprozesses anhand der vorgegebene Software
- Optional: Erweiterung des Editors um weitere Funktionalitäten zur semi-automatischen Erzeugung der Konfigurationsdateien
- Optional: Code-Bibliothek zur direkten Nutzung der Konfigurations-Xml-Datei
- Optional: Untersuchung weiterer Display-Konfigurationen (z.B. gekrümmter Projektionen) und weiterer Visualisierungs- und VR-Software

# Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Arbeit zum Thema:

*Open Display Environment Configuration Language*

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 16. August 2015

Ronald Großmann

## Kurzfassung

Durch die Vielfalt und Heterogenität von Large High Resolution Displays (LHRD) ist es sehr aufwändig Anwendungen für diese zu konfigurieren. Mit Open Display Enviroment Description Language (OpenDECL) stellt die vorliegende Diplomarbeit eine XML-Beschreibung für LHRDs vor. OpenDECL liegt als XML-Spezifikation in einer XML-Schema vor. Mit Hilfe von XSL-Transformationen können aus diesen Beschreibungen die Konfigurationen für Anwendungen generiert werden, welche auf den entsprechenden LHRDs ausgeführt werden. Mit zwei Anwendungen aus dem Bereich der wissenschaftlichen Visualisierung und vier verschiedenen LHRD-Installationen wurde dies erfolgreich getestet. Zum Bearbeiten und Anwenden von OpenDECL wird zusätzlich ein Editor zur Verfügung gestellt. OpenDECL stellt erfolgreich ein Hilfsmittel zur Verfügung um LHRD-Installationen zu Beschreiben und Konfigurationen von Anwendungen dafür zu generieren.



## Abstract

Due to the variety and heterogeneity of Large High Resolution Displays (LHRD) it is very complicated to configure applications for them. With Open Display Enviroment Description Language (OpenDECL), this diploma thesis presents a XML description for LHRDs. OpenDECL is a XML specification and available in a XML schema. With the help of XSL transformations, the configurations can be generated from these descriptions for applications that run on the corresponding LHRDs. This has been successfully tested with two applications from the field of scientific visualization and four different LHRD installations. Additionaly an editor is provided to edit and apply OpenDECL. OpenDECL successfully provides a tool to describe LHRD installations and to generate configuratons for applications that run on them.

# Inhaltsverzeichnis

# 1 Einführung

Ein Tiled Display beziehungsweise Large High Resolution Display (LHRD) im Sinne dieser Arbeit, beschreibt Installationen, bei denen mehrere Displays zu einem größeren Display zusammengesetzt werden. Im Falle von Monitoren, besteht das LHRD aus mehreren einzelnen Monitoren die ein gemeinsames Display darstellen. Dieses muss nicht in einer Ebene liegen, sondern kann auch eine Krümmung aufweisen. Im Falle von Projektoren, besteht das LHRD aus mehreren Projektionsflächen für die Projektoren. Projektoren können auch auf ein und die selbe Fläche projizieren. Dabei können die Projektionen beispielsweise durch Farbfilter für jedes Auge wieder getrennt werden und so eine stereoskopische Abbildung erzeugt werden.
LHRD bieten den Vorteil eine große Pixeldichte auf einer großen Fläche darzustellen. Dadurch lassen sich Details lokal noch hochauflösend darstellen, während gleichzeitig das Gesamtbild und Kontext dargestellt werden kann. Diese Eigenschaften machen LHRD besonders attraktiv für Anwendungsgebiete, in denen viele Informationen gleichzeitig dargestellt werden oder der Kontext einer Information wichtig ist. Ein solches Anwendungsgebiet findet sich in der wissenschaftlichen Visualisierung. Hier lassen sich sich dank der Größe des Displays sehr viele Daten gleichzeitig darstellen. Auch für Kommando und Kontrollaufgaben finden LHRD Anwendung, beispielsweise im Militär oder in der Luftfahrt. Besonders konstruierte LHRD, wie beispielsweise die CAVE [CNSD93], bieten dem Nutzer eine besonders immersive Umgebung. Diese kann in der Industrie zur Unterstützung beim Design benutzt werden, beispielsweise bei der Entwicklung in der Automobilindustrie.
Der Aufbau und Betrieb eines LHRD ist aufwändig, weshalb diese Technologie noch nicht so weit verbreitet ist. Die Komponenten für ein LHRD müssen meist individuell ausgewählt und konfiguriert werden, da es einige Probleme beim Betrieb eines LHRD gibt, auf die im folgenden eingegangen wird.

## 1.1 Probleme

Durch die hohe Anzahl der Pixel in LHRDs ist es nötig mehrere GPUs für die Bilderzeugung zu verwenden um in annehmbarer Zeit ein Bild zu erzeugen. Gerade bei interaktiven Visualisierungen ist eine möglichst kurze Reaktionszeit und ein möglichst flüssiger Ablauf der Visualisierung erforderlich. Für gewöhnlich wird dafür ein Cluster aus mehreren Rechnern zur Bilderzeugung verwendet, wodurch sich weitere organisatorische Probleme ergeben. Zum einem kann die Berechnung von dem, was visualisiert werden soll, getrennt von der eigentlichen Bilderzeugung geschehen. Es gäbe dabei bestimmte Rechner, die beispielsweise für die Simulation oder Rechner die nur für Bilderzeugung und das entsprechende Darstellen auf dem LHRD verantwortlich sind. Die Daten, die dabei von Rechnern der Simulation zu den Rechnern der Bilderzeugung transportiert werden müssen, sind nicht besonders groß, da es sich nur um Geometrieinformationen und Grafikbefehle handelt. Jedoch ist die Zuordnung, welchen Teil der Simulation welcher Rechner für die Bilderzeugung benötigt, damit das entsprechende Display im LHRD richtig angezeigt wird, nicht einfach. Zum anderem gibt es die Möglichkeit das Berechnen und Bilderzeugen auf einem oder mehreren Rechnern durchzuführen und die erzeugten Bilddaten dann an die Rechner zu schicken, die mit dem LHRD verbunden sind. Dabei erhalten die mit dem LHRD verbundenen Rechner nur die Bilddaten, die dem entsprechenden Teil des LHRD abbilden. Das Problem dabei ist die Größe der Bilddaten, die bei ausreichend hoher Bildrate, übertragen werden müssen. Bei beiden dieser Ansätze gibt es das große Problem der Synchronisation. Wenn die Simulation beziehungsweise Berechnung verteilt über mehrere Rechner geschieht, müssen diese untereinander synchronisiert werden um konsistente Daten zu produzieren. Die Bildausgabe auf die einzelnen Displays des LHRD muss ebenfalls synchro-

nisiert geschehen um ein gutes Ergebnis zu erzielen.
Ein weiterer Problembereich für LHRD ist die Kalibrierung und Einrichtung an sich. Gerade bei Projektor-basierenden LHRDs muss sehr genau kalibriert werden, um ein optimales Bild zu erhalten. Da die Geometrie der Projektion aus technischen Gründen nicht perfekt rechtwinkelig sein kann, ist es schwer eine möglichst verzerrungsfreie Abbildung zu erhalten. Wenn man mehrere Projektoren benutzt um ein LHRD zu erzeugen ist die Helligkeit der Projektion ein Problem. Diese ist je nach Projektor ungleich über die Projektionsfläche verteilt und gerade am Rand kommt es zu Problemen, wenn man einen möglichst unauffälligen Übergang von einer in die anliegende Projektion erhalten möchte. An diesen Stellen muss mittels Edge Blending die Helligkeit angepasst werden, sodass ein gleichmäßiger und unauffälliger Übergang zwischen den Projektionen entsteht. Oftmals werden bei solchen Systemen Spiegel benutzt, um die Projektionsstrahlen umzulenken und somit räumlich kompaktere Installationen zu erhalten. Durch die zusätzliche Umlenkung durch Spiegel wird jedoch das Kalibrieren der Projektion schwieriger, da auch die Ausrichtung des Spiegels mit beachtet werden muss. Bei Monitor basierenden LHRD ist es etwas einfacher die Geometrie beziehungsweise Form des LHRD zu erhalten, da es einfacher ist die Monitore in einem regelmäßigen Gitter anzuordnen. Das größte Problem bei solchen LHRDs sind jedoch die Rahmen, die zwangsläufig an jedem Monitor vorhanden sind. Auch wenn es inzwischen Monitor Modelle mit sehr schmalen Rahmen gibt, verhindern diese Rahmen nach wie vor den Eindruck, dass es sich um eine große, ununterbrochene Bildfläche handelt. Auch bei Monitoren gibt es Probleme mit der Helligkeitsverteilung, vor allem am Rand. Diese ist auch bei Monitoren vom gleichen Modell von Exemplar zu Exemplar unterschiedlich.
Ein dritter Problembereich für LHRDs betrifft die Anwendungen, die auf dieses Systemen betrieben werden sollen. Da viele LHRD-Installationen individuell gebaut und betrieben werden, ergibt sich eine sehr heterogene Landschaft aller LHRDs. Dies hat zur Folge, dass Anwendungen oft individuell für einzelne LHRD-Installationen entwickelt werden, beziehungsweise bestehende Anwendungen umfangreich angepasst werden müssen. Dies schränkt die Portabilität dieser Anwendungen ein, da es nur wenige Standards oder portable Frameworks für die Entwicklung gibt.

# 2 Verwandte Arbeiten

In [Mor12] wird die bisherige Forschung und Entwicklung an LHRDs evaluiert und vorgeschlagen, in welche Richtung sich die Forschung auf diesem Gebiet weiter entwickeln sollte. Bei diesen Vorschlägen wird vor allem versucht, den Fokus vom LHRD an sich mehr auf die Anwendungen und den Nutzer dafür zu verlagern. Wie man LHRD konstruiert ist bekannt, doch an sich nicht trivial und mit immer wiederkehrenden Herausforderungen. Deshalb sollte die Industrie stärker in die Produktion von LHRD einsteigen um so diese zu vereinfachen und damit die Entwicklung voran zu bringen. Die Größe von LHRDs bringt neue Herausforderungen bezüglich Interaktion und Wahrnehmung von Informationen mit sich, welche künftige Anwendungen für LHRDs berücksichtigen müssen.
In [NSS+06] wird eine Übersicht über Geräte, Anwendungen und Techniken für LHRD gegeben. Die vorgestellte Hardware reicht von Multi-Monitor und Multi-Projektor-Installationen über Stereo-Displays und Head-Mounted-Displays bis hin zu CAVE und Volumendisplays. Es werden verschiedene Anwendungen für Data-Streaming und Distributed-Rendering vorgestellt. Am Ende wird eine Liste vorgestellt, mit den zehn wichtigsten Forschungszielen für LHRDs.
Wie LHRDs für alltägliche Arbeiten genutzt werden können, wurde anhand von Probanden in [EBZ+12] untersucht. Diese nutzten LHRDs über 10 Monate für ihre täglichen Aufgaben. Anhand dieser Erfahrungen wurden Faktoren ermittelt, die bei der Entwicklung von Arbeitsplätzen mit LHRDs beachtet werden müssen. Das sind vor allem die Höhe und Krümmung des Displays. Aber auch die Position der Eingabegeräte und des Nutzers sind wichtig, insbesondere deren Mobilität vor dem Display.
Das Problem der Maussteuerung auf LHRDs wird in [KBSR07] versucht zu lösen. Aufgrund der Größe und Pixeldichte von LHRDs ist es schwierig mit der Maus gezielt zu arbeiten, da man nicht immer das gesamte Display im Blick haben kann und so den Überblick, wo sich der Mauszeiger aktuell befindet, verliert. Außerdem schränkt die Maussteuerung die Bewegungsfreiheit vor dem LHRD ein. In dieser Arbeit wurde ein System entwickelt, in dem man mit Hilfe eines Laserpointer den Mauszeiger steuern kann. Dabei wird mit mehreren Kameras das LHRD mit einer möglichst hohen Bildrate beobachtet. Die Kameras sind so auf das LHRD kalibriert, dass eine Umrechnung von Kamerakoordinaten in Displaykoordinaten möglich ist. Damit wurde der Laserpunkt von den Kameras verfolgt und der Mauszeiger entsprechend positioniert. In der Evaluation zeigte sich jedoch, dass die Leistung der Benutzer mit diesem System hinter denen mit der Maus liegen.

## 2.1 Betrieb von Large High Resolution Displays

Mit Chromium wird in [HHN+02] ein Framework für interaktives Rendering auf Clustern vorgestellt. Dabei wird ein Stream aus Grafik-API-Befehlen (zum Beispiel OpenGL) durch die einzelnen Rechner im Cluster manipuliert. Diese können mehrere Streams empfangen, diese verarbeiten und manipulieren und dann an mehrere nachfolgende Rechner des Clusters als Stream wieder ausgeben. Zur Manipulation des Streams besitzt jeder Rechner im Cluster eine OpenGL-Stream-Processing-Unit, die mit verschiedenen Funktionen den Inhalt des Stream transformiert. Weitere Teile von Chromium betreffen die Kommunikation innerhalb des Clusters, zum Beispiel das Ver- und Entpacken von Streamdaten oder die Abstraktion des Netzwerkes damit eine Kommunikation zwischen verschiedenen Stream-Processing-Units stattfinden kann. All dies ermöglicht, dass der Algorithmus einer Anwendung die mit Chromium benutzt wird, nicht verändert werden muss, da Chromium die Ausgabe der Anwendung als Stream übernimmt und die Ausgabe auf ein LHRD ausführt. Die nötigen Korrekturen und Transformationen geschehen im Cluster.
In [JJR+] wird mit SAGE (Scalable-Adaptive-Graphics-Environment) eine Schnittstelle zwischen Appli-

kation und LHRD vorgestellt. Mit SAGE ist es möglich mehrere Bilderzeugungsquellen, beispielsweise Simulationen oder Visualisierungen, zusammenzuführen und auf einem LHRD auszugeben. Die Anzeige der einzelnen Quellen auf dem LHRD lassen sich beliebig skalieren und positionieren. Damit die Quellen mit SAGE verarbeitet werden können, müssen diese für das SAGE-Framework angepasst werden. Dies besteht darin, dass bestimmte Befehle für die SAGE-Application-Interface-Library hinzugefügt werden. Diese regelt die Kommunikation und den Transport des Outputs der Applikation an SAGE.
In [CCL+01] werden zwei Ansätze vorgestellt, wie Anwendungen auf Cluster betriebenen Displays betrieben werden können. Der erste Ansatz beschreibt eine Master-Slave-Architektur, bei der die Ausgabe auf die für die Bilderzeugung zuständigen Rechner verteilt wird. Für die Umsetzung dieses Ansatzes werden zwei Vorschläge gemacht. Zum einem ein spezieller Treiber für die GPU, welcher die Primitive und Befehle, die an die GPU geschickt werden, abfängt und an die Rechner für die Bilderzeugung weiterschickt. Zum anderem eine spezielle *OpenGL32.dll*, welche die Befehle und Daten zur Bilderzeugung nicht an die GPU, sondern an die Rechner für die Bilderzeugung überträgt. Dabei müsste jedoch die entsprechende Anwendung an diese *OpenGL32.dll* angepasst werden. Der zweite Ansatz ist der, des synchronen Ausführen des Programmes. Dabei wird angenommen, dass ein und die selbe Anwendung auf den Rechnern für die Bilderzeugung synchron ausgeführt wird. Für die Umsetzung dieses Ansatzes werden auch zwei Vorschläge gemacht. Zum einem eine Synchronisation des Systems, in der die Anwendung ausgeführt wird, mit den Systemen auf den anderen Rechnern. Dafür müssen die Systeme untereinander Informationen über ihren Status austauschen, um synchron zu bleiben. Zum Anderem eine Synchronisation der Anwendung selbst mit den Anwendungen auf den anderen Rechnern. Auch hier müssen Informationen über den Status ausgetauscht werden, was hier auf der Ebene der Anwendung geschieht. Für diesen Vorschlag müsste die Anwendung entsprechend angepasst werden.

## 2.2 Kalibrierung von Large High Resolution Displays

PixelFlex [YGH+01] ist ein Projektor-basierendes LHRD, welches aus bis zu acht Projektoren besteht. Diese projizieren über individuell neig- und schwenkbare Spiegel auf eine Projektionsfläche. Die Spiegel sowie die Fokussier- und Zoomfunktionen der Projektoren sind von einem Rechner steuerbar. Zusätzlich gibt es eine Kamera zur Kalibrierung. Ziel des PixelFlex Systems ist es, automatisch eine Abbildung von Projektorkoordinaten in Weltkoordinaten zu finden, so dass eine verzerrungsfreie und gleichmäßig helle Projektion von allen Projektoren auf die Projektionsfläche stattfindet. Dies geschieht über die Kamera, wobei zunächst die Abbildung von Weltkoordinaten in Kamerakoordinaten ermittelt wird. Über diese wird dann die Abbildung von Weltkoordinaten in Projektorkoordinaten ermittelt für jeden einzelnen Projektor. Mit dieser Abbildung lassen sich die Ausgaben an die Projektoren so konfigurieren, dass ein LHRD entsteht. Zur Anpassung der Helligkeit an den Übergängen, wo sich mehrere Projektionen überlappen, werden Alphamasken benutzt. Diese blenden softwareseitig Teile des Bildes für entsprechende Projektoren aus, sodass die Projektionen nicht mehr überlappen und somit eine gleichmäßige Helligkeit über das LHRD erreicht wird.
Für die Kalibrierung eines Projektor-basierenden LHRD mit einer gekrümmten Projektionsfläche wird in [SM10] ein System vorgestellt. Die Kalibrierung erfolgt mit Hilfe einer Kamera, die auf einer schwenk- und neigbaren Plattform installiert ist. Dabei nimmt die Kamera in verschiedenen Stellungen Bilder der gekrümmten Projektionsfläche auf, wobei auf ausreichender Überlappung der Bilder zueinander zu achten ist. Zusätzlich werden von den verschiedenen Projektoren spezielle Bilder zur Kalibrierung projiziert. Mit diesen Daten können für jeden Projektor entsprechend Korrekturen und Verzerrungen berechnet werden, die eine optimale Darstellung des LHRD auf der gekrümmten Projektionsfläche ermöglichen.
In [CCF+00] wird die Kalibrierung eines Projektor-basierenden LHRD mit Hilfe einer Kamera vorgestellt, der ohne die Kalibrierung der Kamera funktioniert. Statt einer Abbildung der Weltkoordinaten der Projektionsfläche über die Kamerakoordinaten in die Projektorkoordinaten zu finden, geschieht dies hier über einen heuristische Optimierung mittels Simulated Annealing. Das Ziel dieser Optimierung ist eine Abbildung zu finden, welche den Abstand zweier Punkte, die von verschiedenen Projektoren projiziert

werden, minimiert. Dabei wird angenommen, dass die Punkte die gleiche Stellen auf der Projektionsfläche beschreiben.
Welche Probleme beim Bau eines Projektor-basierenden LHRD mit stereoskopischer Abbildung auftreten und deren mögliche Lösungen werden in [BGA+03] vorgestellt. Das LHRD dieser Arbeit bestand aus einem drei mal vier Gitter von Projektionsflächen, wobei jede Projektionsfläche von jeweils zwei Projektoren bestahlt wurde. Da eine perfekte rechteckige Projektion aufgrund von leichten Verzerrungen, die bei jedem Projektor auftreten, nicht möglich war, mussten verschiedenen Maßnahmen ergriffen werden. Für die grobe Kalibrierung wurden die Projektoren mechanisch positioniert und orientiert. Bei stereoskopischen Abbildungen, wie in diesem Fall, gibt es zusätzlich das Problem, dass für ein optimales Ergebnis, beide Projektionen von ein und der selben Position ausgehen müssen, was physikalisch nicht möglich ist. Dies wird versucht mit Hilfe von speziellen Projektoren zu lösen, die eine off-axis Projektion mit Hilfe von Lens-Shifting ermöglichen. Die feine Kalibrierung der Projektionsgeometrie erfolgt dann softwareseitig mithilfe von Verzerrungen, sodass man für jeden Projektor eine horizontale und vertikale linientreue Projektion erhält, wobei die Projektionen der Paare von Projektoren, die auf ein und die selbe Projektionsfläche projizieren, deckungsgleich sind. Für den Übergang zwischen den sich überlappenden Projektionsflächen wurde Edge Blending mithilfe einer softwareseitigen Alphamaske verwendet. Eine hardwareseitige Lösung mittels spezieller Blenden, die an den Projektoren installiert werden, wurde auch in Betracht gezogen.
Mit XMegaWall wird in [KC07] ein Projektor-basierendes LHRD vorgestellt, welches aus 28 Projektoren besteht, die in einem sieben mal vier Gitter angeordnet sind. Alle Projektoren stehen dabei in einem Gerüst, wobei jeder einzelne Projektor auf einer speziellen Halterung montiert ist. Diese Halterung erlaubt die mechanische Feinjustierung der einzelnen Projektoren.
Für nicht planare Projektor basierende LHRDs wird in [Mor12] eine Methode für eine automatische Kalibrierung mit Hilfe einer Kamera vorgestellt. Dabei wird die gesamte Projektionsfläche des LHRD als Translationsfläche bestehend aus einem Profil und einem Pfad betrachtet. Über zwei nichtlineare Optimierungen werden zunächst die Kameraparameter bestimmt um damit anschließend die dreidimensionale Form der Projektionsfläche zu rekonstruieren. Dabei kann die Kamera auf einer schwenk- und neigbaren Plattform gedreht werden um das gesamte LHRD zu erfassen. Mit Hilfe eines projizierten Musters jedes Projektors, kann nun eine Abbildung von Projektorkoordinaten in Weltkoordinaten berechnet werden und somit die Projektion auf die Projektionsfläche entsprechend angepasst werden.
Für Projektoren mit Weitwinkel Linsen wird in [JGS+07] ein Algorithmus vorgestellt, mit dem sich Verzerrungen durch Linse und nicht planarer Projektionsfläche korrigieren lassen. Der Vorteil bei Projektoren mit Weitwinkel Linsen liegt darin, dass auch mit einem geringen Abstand zur Projektionsfläche eine große Projektion erreicht werden kann. Jedoch erzeugen diese Weitwinkel Linsen erhebliche Verzerrungen. Der in dieser Arbeit vorgestellte Algorithmus ermittelt zunächst die geometrische Form der Projektionsfläche. Dies geschieht mittels strukturiertem Licht des Projektors und einer kalibrierten Stereokamera. Anschließend wird die Verzerrung der Linse ermittelt und zusammen mit der geometrischen Form der Projektionsfläche eine Vorverzerrung des Bildes für den Projektor berechnet, damit eine optimale Projektion auf die Projektionsfläche stattfindet.
In [SM11] wird eine Technik zur Kalibrierung von Projektor-basierenden LHRD mit einer Kamera beschrieben. Unter den Bedingungen, dass die Projektionsfläche vertikal extrudiert ist, dass das Seitenverhältnis des Rechteckes, welches die vier Ecken der Projektionsfläche aufspannen, bekannt ist und die Grenze der Projektionsfläche sichtbar sowie segmentierbar ist, kann dies mit einer nicht kalibrierten Kamera geschehen. Zunächst wird mit Hilfe eines Bildes der Projektionsfläche die Kameraposition und Orientierung sowie die dreidimensionale Geometrie der Projektionsfläche ermittelt. Dann wird für jeden Projektor anhand eines projizierten Musters dessen Parameter bestimmt und damit eine Abbildung von Projektorkoordinaten in Weltkoordinaten auf der Projektionsfläche bestimmt.

## 2.3 Middleware für Anwendungen auf Large High Resolution Displays

In [RFC+03] wird ein Ansatz vorgestellt, wie verschiedene heterogene Applikationen zu einer VR-Anwendung verknüpft werden können. Dies geschieht, in dem man die einzelnen Applikationen zu Modulen kapselt, die nur noch über entsprechend spezifizierte input und output ports kommunizieren. Die Kommunikation erfolgt über Messages in denen die Daten zur Weiterverarbeitung an das entsprechende Modul gesendet werden. Dieser Ansatz einer Message Oriented Middleware bietet den Vorteil, dass die einzelnen Module, austauschbar sind, solange die input und output ports gleich spezifiziert sind. Der Message Manager der Middleware steuert dann den Event basierenden Ablauf einer Anwendung mit Datenverteilung und Lastbalancierung. Die Bilderzeugung und Konfiguration erfolgt in den Modulen.
Einen ähnlichen Ansatz verfolgt FlowVR [AGL+04] welches eine Middleware darstellt die vor allem für verteilte VR-Anwendungen auf Cluster konzipiert ist. Auch hier sollen heterogene Einzelkomponenten zu einer gesamt VR-Anwendung verknüpft werden, ohne den bestehenden Code der Einzelkomponenten zu stark zu verändern. Das Hauptaugenmerk liegt dabei auf der Kommunikation und Synchronisation innerhalb des Clusters. Ähnlich zu [RFC+03] haben die einzelnen Module input und output ports, die mit anderen Modulen verbunden sind. Die Kommunikation läuft auch hier über Messages. Dabei gibt es spezielle Messsages zur Synchronisation, womit beispielsweise eine gleichmäßige Bildrate für die Ausgabe auf ein LHRD erzielt wird, wenn dieses LHRD von mehreren Rechnern des Cluster betrieben wird.
Im Falle der CAVE [CNSD+92] handelt es sich um ein besonderes LHRD. Da der Nutzer sich in einem Raum befindet, der von mehreren Seiten projiziert wird, bekommt der Nutzer eine besonders immersive Erfahrung. Die Entwicklung von VR-Anwendungen für solche Installation ist nicht einfach, da es eine Vielzahl an Ein- und Ausgabegeräten gibt, für die die Anwendung angepasst werden muss.
In [BJH+01] wird mit dem VR Juggler eine virtuelle Platform zur Entwicklung und Ausführung von VR-Anwendungen in CAVE Installationen vorgestellt. Das Ziel hierbei war, den Entwicklern der VR-Anwendungen einfache Software Werkzeuge zu geben und gleichzeitig die Komplexität der Hardware Konfiguration weg zu kapseln. Damit sollen sich Entwickler voll auf die eigentliche VR-Anwendung konzentrieren können und eine einfache Basis zum Entwickeln, Testen und Ausführen haben. Dabei agiert VR-Juggler ähnlich der virtuellen Maschine bei JAVA, wie eine zusätzliche Schnittstelle zwischen Anwendung und Hardware. Dadurch entstehen nur geringe oder nicht messbare Leistungsnachteile. Eine Anwendung, die einmal für den VR-Juggler entwickelt wurde, läuft also auf allen Installationen, die VR-Juggler unterstützt.
Mit HIVE wird in [WJS04] und [WSJ06] ein Middleware Framework vorgestellt für die Entwicklung verteilter VR-Anwendungen. Besonderes Augenmerk wurde dabei auf die Skalierbarkeit bezüglich des Clusters, die Effizienz der Server und eine offene Plattform gelegt. Dabei arbeitet im Hintergrund ein System aus drei Schichten. In der untersten Schicht sind die Nutzer, die aktiv oder passiv in einer virtuellen Umgebung teilnehmen kann. In der mittleren Schicht agiert ein Group Manager Prozess, der die virtuelle Umgebung spezifiziert und die Nutzer verwaltet. In der obersten Schicht agiert ein Group Agent Prozess, der die verschiedenen Group Manager Prozesse verwaltet, sowie ein Directory Manager Prozess, der die Kommunikation, das Lastbalancieren der Nutzer und damit die Wurzel des Systems ist.
Mit InTml wird in [FGH02] eine Beschreibung für VR-Anwendungen vorgestellt. Dabei werden verschiedene Eingabegeräte, Ausgabegeräte und Interaktionstechniken beschrieben und zusammen mit entsprechenden Geometriedaten zu Anwendungen verknüpft. Das Konzept besteht aus einem Datenflussmodell, bei dem die verschiedenen Elemente über Inputports und Outputports verbunden sind und die Daten verarbeiten und weitergeben. Die Beschreibung der einzelnen Elemente liegt dabei als XML Datei vor, in dem beschrieben wird, was dieses Element macht und welche Daten es verarbeiten kann und welche es anschließend weiter gibt. Die Verbindungen zwischen den einzelnen Elementen wird ebenfalls in einer XML Datei beschrieben.
Vrui VR Toolkit [Kre] versucht ein Tool-Kit für die Entwicklung von VR-Anwendungen zur Verfügung zu stellen. Dabei wird versucht, die Ausgabe auf Displays, die verteilte Berechnung in einem Cluster und

die Eingabe mit verschiedenen Eingabegeräten so gut es geht zu abstrahieren. Damit sollen Entwickler entlastet werden, da sie sich nicht mehr um die spezielle Implementierung dieser Aspekte für konkrete Installationen kümmern müssen. Die Anwendungen werden für diese abstrahierten Aspekte entwickelt. Um diese Anwendungen dann auf verschiedenen Installationen zu betreiben, werden mit Konfigurationsdateien die konkreten Hardwareelemente beschrieben und mit Vrui in die abstrahierten Aspekte übersetzt. Somit kann man ein und die selbe VR-Anwendung auf verschiedenen Installationen betreiben und muss dafür lediglich die Konfigurationen anpassen.

## 2.4 Einordnung meiner Arbeit

Diese Arbeit befasst sich mit dem Thema der Kalibrierung von Anwendungen für LHRDs. Es stellt ein Hilfsmittel für Anwendungen auf LHRDs dar. Dabei verfolgt die Arbeit einen Ansatz der in den von mir recherchierten Arbeiten noch nicht behandelt wurde. Diese Arbeit konzentriert sich dabei auf die Konfiguration und die Anordnung des LHRD an sich. Die Konfiguration der Infrastruktur zum Betrieb eines LHRD steht dabei eher im Hintergrund, wird jedoch auch mit aufgegriffen. Diese Arbeit stellt ein Hilfsmittel zur Verfügung, mit der bestehende Anwendungen auf Basis einer Beschreibung des LHRD, individuell konfiguriert werden können.

# 3 Anforderungsanalyse

Die Herausforderung der Aufgabenstellung bestand darin, die verschiedensten LHRD Installationen auf die essentiellen Aspekte zu reduzieren und damit eine gemeinsame abstrakte Beschreibung zu finden. Dabei sollten nur die wichtigsten Aspekte teil der Beschreibung sein, jedoch genügend, um die Konfiguration der Anwendungen zu erstellen. Ziel ist ein Standard, der in der Lage ist, jedes LHRD zu beschreiben.

Im Zuge der Anforderungsanalyse, habe ich die in der Aufgabenstellung genannte Hard- und Software analysiert. Hierbei habe ich mich im Falle der Hardware vor allem auf die Anordnung und Zusammensetzung der Displayelemente konzentriert, welche das LHRD bilden. Dabei ging es einerseits um die rein physischen Maße und Gegebenheiten, aber auch die Infrastruktur dahinter. Wie die Displayelemente angesteuert und abgebildet werden ist ein wichtiger Aspekt.
In der Softwareanalyse habe ich mich auf die Konfiguration der Anwendungen konzentriert. Insbesondere die Konfigurierbarkeit der Bildausgabe war für meine Arbeit wichtig. Dies betraf die genaue Positionierung der Ausgabefenster als auch die Abbildung aus der Anwendung auf dieses Ausgabefenster. Ich habe jede der zu unterstützenden Softwarepakete auf den verschiedenen Hardwareinstallationen untersucht.

## 3.1 Hardwareanalyse

Im folgenden werden die verschiedenen Hardware Installationen beschrieben, welche ich in meiner Arbeit analysiert habe. Für die vier Installationen gibt es eine schematische Abbildung (Abb. 3.1, Abb. 3.2, Abb. 3.3, Abb. 3.4) welche auf der linken Seite die Infrastruktur darstellt und auf der rechten Seite den Aufbau des entsprechenden LHRD.

### Desktop Systeme

Eine der Installationen ist ein Desktop System mit mindestens zwei Displays, die nicht zwangsläufig in einer Ebene liegen müssen. Grundsätzlich betrachte ich ein solches Desktop System als LHRD, wenn der Desktop auf die Displays erweitert wird. Dabei hat jedes Pixel der Displays eine eindeutige Desktop Pixel Koordinate. Das Referenz System, was mir zur Verfügung stand, bestand aus einem Notebook mit einem LCD Display. Dieses hatte eine Auflösung von 1280 x 800 Pixel, sowie eine physikalische Größe von 331 x 207 mm. An dieses Notebook ist ein LED Monitor angeschlossen und steht rechts neben dem Notebook, so dass die Oberkanten beider Displays auf gleicher Höhe sind. Der Monitor hat eine Auflösung von 1920 x 1080 Pixel, sowie eine physikalische Größe von 464 x 260 mm (Abb. 3.1). Da beide Displays einen Rahmen haben, können sie nicht lückenlos nebeneinander stehen sondern haben einen Abstand von 28 mm.
Als Betriebssystem lag Windows 7 vor, welches so eingerichtet wurde, dass der LCD Bildschirm des Notebooks der Hauptbildschirm war und somit in dessen oberen linken Ecke der Ursprung des Pixelkoordinatensystem lag. Der Desktop wurde dann nach rechts auf den LED Monitor erweitert. Daraus folgte, dass die linke obere Ecke des LED Monitors die Pixelkoordinaten (1280;0) hatte.

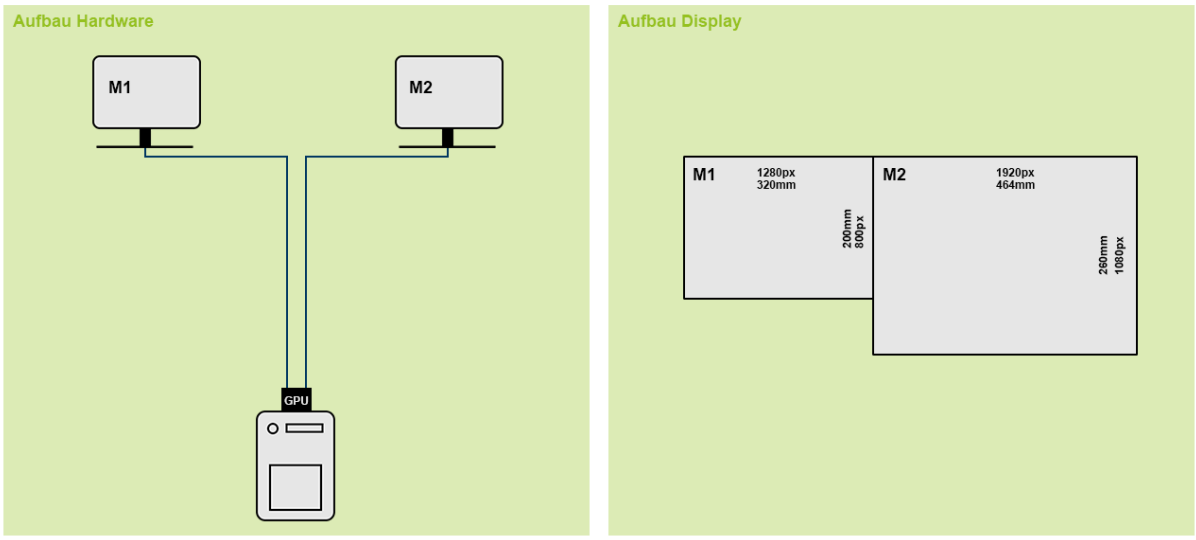


Abbildung 3.1: Schematische Darstellung des Desktop Systems

## CGV Stereowall

Bei der Stereopowerwall der Professur für Computergraphik und Visualisierung handelt es sich um ein Projektor-basierendes LHRD. Dabei sind zwei Projektoren an eine GPU eines Rechners angeschlossen (Abb. 3.2 links). Das Bildsignal an einen der beiden Projektoren wird noch einmal gedoppelt um es an einem Monitor anzuzeigen (Abb. 3.2 P1 und M). Dieser Monitor dient zur vereinfachten Bedienung des Rechners. Die Besonderheit dieser Installation besteht darin, dass beide Projektoren auf ein und die selbe Projektionsfläche strahlen, und die Projektionen für eine stereoskopische Abbildung möglichst deckungsgleich aufeinander liegen (Abb. 3.2 rechts). Dazu wurden die beiden Projektoren direkt übereinander montiert und entsprechend eingestellt. Um Platz zu sparen werden die Strahlen über einen Spiegel umgelenkt, bevor sie von Hinten auf die Projektionsfläche treffen. Vor beiden Projektoren sind Polarisationsfilter so montiert, dass sich die Polarisation der Strahlen beider Projektoren um 90° unterscheidet. Die Projektionsfläche besteht aus einem Stoff, der diese Polarisation erhält und somit zusammen mit einer entsprechenden Stereobrille für Polarisation ein getrenntes Bild für das linke und rechte Auge darstellt. Die Projektionsfläche hat eine Größe von 2720 x 2040 mm. Die beiden Projektoren haben eine Auflösung von 1400 x 1050 Pixel.
Der Rechner an den die Projektoren und der Monitor angeschlossen waren, hatte unter anderem Windows 7 als Betriebssystem, welches ich genutzt habe. Dabei war der Desktop über beide Projektoren erweitert, die Gesamtgröße des Desktops betrug also 2800 x 1050 Pixel. Dabei war der Projektor, der für das linke Auge projizierte, auch der linke Teil des Desktops. Dies war auch der Teil, der auf dem zusätzlichen Monitor angezeigt wurde. Dementsprechend wurde der rechte Teil des Desktop von dem Projektor dargestellt, welche für das rechte Auge projizierte.

## MT Powerwall

Bei der Powerwall des Lehrstuhl für Multimedia-Technologie handelt es sich um ein LHRD bestehend aus zwölf Monitoren die in einem vier mal drei Gitter angeordnet sind (Abb. 3.3 rechts). Jeder dieser Monitore hat eine Auflösung von 1920 x 1080 Pixel, sowie eine physikalische Größe von 1210 x 680

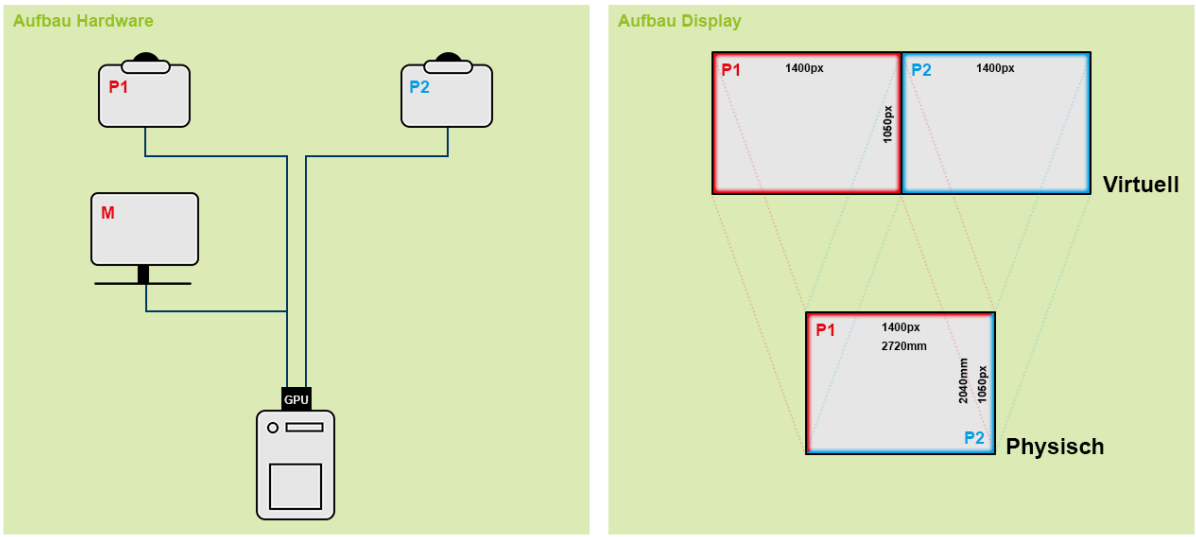


Abbildung 3.2: Schematische Darstellung der Stereopowerwall der Professur für Computergraphik und Visualisierung

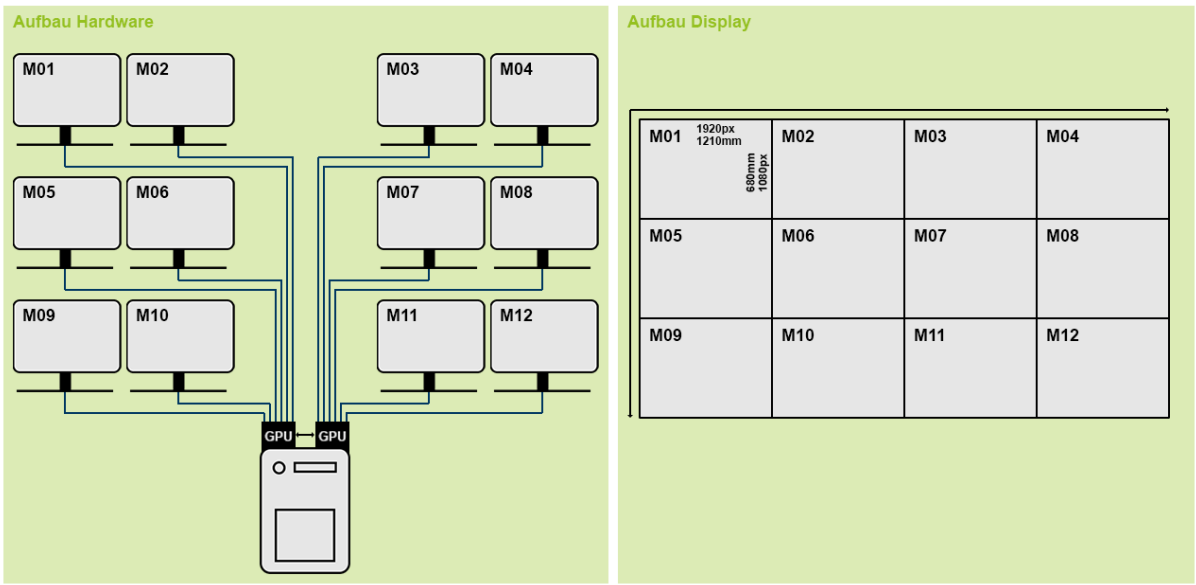


Abbildung 3.3: Schematische Darstellung der Powerwall des Lehrstuhl für Multimedia-Technologie

mm. Das gesamte LHRD hat also eine Auflösung von 7680 x 3240 Pixel, sowie eine metrische Größe von 4855 mm horizontal und 2050 mm vertikal. Die Rahmen um die Monitore sind mit 2,5 mm äußerst schmal.
All diese Monitore werden von einem Rechner mit zwei GPUs betrieben. An jeder GPU sind sechs Monitore angeschlossen (Abb. 3.3 links). Auch auf diesem Rechner stand unter anderem Windows 7 als Betriebssystem zur Verfügung, welches ich benutzt habe. Der Desktop war über alle zwölf Monitore erweitert mit dem Monitor in der linken oberen Ecke als Hauptmonitor. Dadurch lag der Ursprung des Pixelkoordinatensystem für den gesamten Desktop auch in der linken oberen Ecke.
Trotz der zwei leistungsstarken GPUs, hatte das System unter Windows manchmal Probleme mit der großen Pixelzahl. Gerade bei Testen der Softwarepakete kam es des öfteren zu Abstürzen oder Bildfehler für einige Monitore, welche den Test erschwerten.

## CAVE

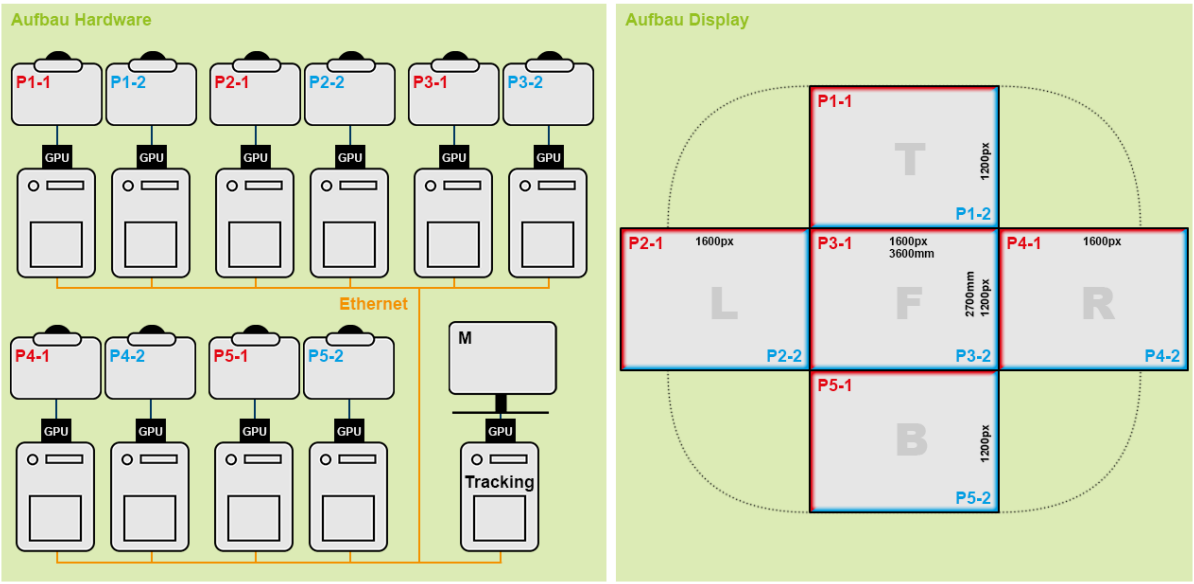


Abbildung 3.4: Schematische Darstellung der Fünf-Seiten-CAVE des Lehrstuhl Konstruktionstechnik/CAD

Bei der CAVE des Lehrstuhl Konstruktionstechnik/CAD handelt es sich um ein Projektor-basierendes LHRD. In diesem Fall handelt es sich um eine Fünf-Seiten-CAVE, bei der nur die Seite nicht für die Projektion benutzt wird, durch die man die CAVE betritt. Die restlichen fünf Seiten werden von hinten jeweils von zwei Projektoren deckungsgleich bestrahlt. Jede der Seiten hat eine Größe von 3600x 2700 mm (Abb. 3.4 rechts). Da die Seiten nicht quadratisch sondern rechteckig mit einem Verhältnis von vier zu drei sind, kommt es zu einer Diskrepanz zwischen den Seiten links und rechts (Abb. 3.4 rechts L und R) und den oberen und unteren Seiten (Abb. 3.4 rechts T und B). Dadurch reichen die Seiten oben und unten nicht ganz bis zum Eingang der CAVE. Jeder der zehn Projektoren hat eine Auflösung von 1600 x 1200 Pixel. Im Gegensatz zur Stereopowerwall der Professur für Computergraphik und Visualisierung, werden hier die Bilder für linkes und rechtes Auge durch einen Interferenzfilter getrennt. Dabei werden mehrere Wellenlängen der Strahlen jeweils gefiltert, sodass mit einer entsprechenden Brille die Bilder für die Augen wieder getrennt wahrgenommen werden. Auch hier werden mittels Spiegel die Stahlen umgelenkt um Platz zu sparen.

Jeder der zehn Projektoren ist mit jeweils einem separaten Rechner verbunden. Diese zehn Rechner und ein zusätzlicher Rechner sind durch ein Ethernet Netzwerk verbunden (Abb. 3.4 links). Der zusätzliche Rechner hat einen eigenen Monitor außerhalb der CAVE. Dieser spezielle Rechner dient zum Steuern der anderen 10 Rechner via Remote Control. Zusätzlich verarbeitet dieser Rechner die Tracking Daten und sendet sie an die 10 Rechner mit den Projektoren. Auf allen Rechnern ist Windows XP als Betriebssystem installiert. Aufgrund der speziellen Treibersoftware für die GPUs und der Software zum betreiben der CAVE, konnte noch nicht auf ein aktuelleres Betriebssystem umgestiegen werden.

## 3.2 Softwareanalyse

In der Softwareanalyse habe ich mich auf die Konfigurierbarkeit der einzelnen Anwendungen konzentriert. Dies betrifft hauptsächlich die Konfiguration der Bildausgabe. Dabei geht es einerseits um die Positionierung der Ausgabe, also wie kann ich das Fenster der Ausgabe den Anforderungen entsprechend positionieren. Auch die Anpassung des Viewports, zum Beispiel bezüglich des FOV oder der Blickrichtung, sollte konfigurierbar sein. Bei Stereoprojektionen ist es zudem wichtig, die einzelnen Ausgaben für linkes und rechtes Auge trennen zu können.

### MegaMol

Bei MegaMol handelt es sich um ein Softwaresystem zur Visualisierung von Punkt-basierenden Datensätzen [GKM+15].
MegaMol-Projekte bestehen aus einem Graph verschiedener Berechnungsmodule, die auf den Datensatz angewandt werden. Dabei können Projekte auch in mehreren Instanzen gestartet werden, welche dann jeweils auch ein eigenes Ausgabefenster besitzen. Für das Rendern auf mehrere verteilte Displays bietet MegaMol ein konfigurierbares TileView-Modul an. In diesem lassen sich auch die Stereoeigenschaften konfigurieren.
Die Konfiguration erfolgt an zwei Stellen. Zum einem in der Datei *megamol.cfg* wo für verschiedene Fenster die Größe und Position festgelegt werden kann. Der Name des Fensters ist dabei der Name der Instanz, den man beim Starten des Projektes angibt, gefolgt von "-window" (Abb. 3.5). Die Werte im *value* Attribut stehen dabei der Reihenfolge nach für die x- und y-Koordinaten der linken oberen Fensterecke sowie die Breite und Höhe des Fensters in Pixel. Die Konfiguration der TileView erfolgt dann über eine Parameter-Datei, die beim Starten des Projektes mit übergeben wird. Darin werden die Eigenschaften der einzelnen Module für jede Instanz gespeichert (Abb. 3.6). Für TileView sind das die Stereoeigenschaften (Abb. 3.6 erste und zweite beziehungsweise fünfte und sechste Zeile), Die Position und Größe des Tiles innerhalb des Viewports (Abb. 3.6 dritte beziehungsweise siebte Zeile) sowie die Gesamtgröße des Viewports (Abb. 3.6 vierte beziehungsweise achte Zeile). Dabei sind die Angaben der Position und Größe nicht an eine bestimmte Einheit gebunden. Mit diesen Konfigurationen lassen sich beliebig Tiles positionieren und anpassen.
Mit diesen Konfigurationen habe ich MegaMol erfolgreich auf dem Desktop System und der Powerwall des Lehrstuhl für Multimedia-Technologie getestet. Auf der Stereopowerwall der Professur für Computergraphik und Visualisierung habe ich MegaMol auch erfolgreich mit der Stereoprojektion getestet. Da MegaMol noch kein Modul besitzt, mit dem man mehrere Viewports erzeugen kann, die verschiedene Blickrichtungen haben, war MegaMol ungeeignet für einen Test in der CAVE des Lehrstuhl Konstruktionstechnik/CAD.

```
<set name="tile1-window" value="x0y0w1400h1050nd" />
<set name="tile2-window" value="x1400y0w1400h1050nd" />
```

Abbildung 3.5: Konfiguration der Ausgabefenster in *megamol.cfg*

```
::tile1::TileView1::eye=Left Eye
::tile1::TileView1::projType=Stereo OffAxis
::tile1::TileView1::tile=0.000000;0.000000;1400.000000;1050.000000
::tile1::TileView1::virtSize=1400.000000;1050.000000
::tile2::TileView1::eye=Right Eye
::tile2::TileView1::projType=Stereo OffAxis
::tile2::TileView1::tile=0.000000;0.000000;1400.000000;1050.000000
::tile2::TileView1::virtSize=1400.000000;1050.000000
```

Abbildung 3.6: Konfiguration der Tile-Views in Parameter Datei

### ParaView

ParaView [AGL05] ist eine open-source Plattform auf Basis von VTK [SLM04] für die Analyse und Visualisierung von Daten. Die von mir benutzte Version von ParaView ist die vor kompilierte Desktop-Version 4.3 für Windows. Diese besitzt ein auf QT basierenden Interface.
Leider lässt sich die Ausgabe in diesem Interface nicht konfigurieren. ParaView bietet jedoch die Möglichkeit für Remote Rendering. Dafür wird ein ParaView Render Server mit Hilfe von MPI gestartet, mit dem sich dann der ParaView Client verbinden kann. Dies funktioniert, auch wenn der Server auf dem gleichen Rechner wie der Client läuft, mittels *localhost*. Beim Starten des Servers kann man diesen mittels Startparameter konfigurieren (Abb. 3.7). Dabei gibt es die Möglichkeit, die Ausgabe auf dem Rechner des Servers zu erzeugen. Diese Ausgabe lässt sich dann mittels einer Konfigurationsdatei genauer einstellen. Die Konfigurationsdatei wird auch als Startparameter beim Starten des Servers übergeben und ist XML Formatiert. Diese PVX-Dateien (Abb. 3.8) konfigurieren die Ausgabe für die Render Server. *Machine* beschreibt den Rechner auf dem der Server läuft. Dabei ist das Attribute *Name* der Name des Rechners. Im Falle eines Clusters, lassen sich also auch die Ausgaben auf mehreren Rechnern konfigurieren. Die weiteren Attribute konfigurieren die Ausgabe. *Geometry* konfiguriert die Größe und Position des Ausgabefensters. Dabei sind die ersten beiden Werte die Größe des Fensters in Pixel und die letzten beiden Werte die x- und y-Koordinate der linken oberen Fensterecke. *LowerLeft, LowerRight* und *UpperRight* sind drei Raumvektoren, die die entsprechenden Ecken des Displays im Raum beschreiben. Das hierbei angenommene Koordinatensystem ist das Weltkoordinatensystem in ParaView. Um mehrere Ausgabefenster auf dem gleichen Rechner zu erzeugen, genügt ein weiteres *Machine* Element mit dem gleichen *Name* Attribut. Für jedes Ausgabefenster muss jedoch beim Starten des Servers wie MPI mindestens ein Prozess zur Verfügung stehen (Abb. 3.7 Startparameter *-np 2* für zwei Prozesse).
Mit diesen ParaView Renderserver habe ich Paraview erfolgreich auf dem Desktop System und der Powerwall des Lehrstuhl für Multimedia-Technologie getestet. Leider lässt sich nie das ganze LHRD zur Ausgabe benutzen, da die Interaktion mit Paraview weiterhin nur über den Client funktioniert. Somit muss dieser immer mit angezeigt werden. Da ParaView keine Konfiguration zur Verfügung stellt, mit der man Stereoprojektion in separaten Fenster ausgeben kann, habe ich auf einen Test auf der Stereopowerwall der Professur für Computergraphik und Visualisierung verzichtet.
Durch die Möglichkeit des Remote Rendering auf einem Cluster sollte ParaView auch in der CAVE des Lehrstuhl Konstruktionstechnik/CAD laufen. Dabei werden auf den Rechnern des Clusters entsprechend die ParaView-Server mit den entsprechenden Startparametern gestartet. Die durch die PVX Konfigurationsdateien gegebene Möglichkeit mehrere Viewports mit verschiedenen Blickrichtungen zu konfigurieren, ermöglicht eine immersive Visualisierung in der CAVE. Die Kommunikation zwischen den Servern im Cluster wird mittel MPI ermöglicht. Leider konnte ich ParaView nicht erfolgreich in der CAVE des Lehrstuhl Konstruktionstechnik/CAD ausführen und testen. Mir ist es nicht gelungen MPI so auszuführen und zu testen, dass es die ParaView Render Server auf dem Cluster ausführt. Unter Windows war es mir nicht möglich mittels MPI einen Prozess auf einem anderen Windows Rechner zu starten, der sich im gleichen Netzwek befand. Mir ist es nicht gelungen Windows so zu konfigurieren, dass das Starten von Prozessen via Remote erlaubt wurde. Deshalb sind die Konfigurationen von ParaView bezüglich der CAVE des Lehrstuhl Konstruktionstechnik/CAD nur theoretischer Natur.

```
mpirun -np 2 pvserver.exe -display localhost:0 datei.pvx
```

Abbildung 3.7: Beispielbefehl zum Starten des ParaView Servers mit Startparametern

```
<?xml version="1.0" ?>
<pvx>
<Process Type="client" />
<Process Type="render-server">
        <Machine Name="Links"
             Environment="DISPLAY=:0"
             Geometry="1600x1200+0+0"
             FullScreen="0"
             ShowBorders="0"
             LowerLeft="-1.8␣-1.35␣1.8"
             LowerRight="-1.8␣-1.35␣-1.8"
             UpperRight="-1.8␣1.35␣-1.8">
        </Machine>
        <Machine Name="Vorn"
             Environment="DISPLAY=:0"
             Geometry="1600x1200+0+0"
             FullScreen="0"
             ShowBorders="0"
             LowerLeft="-1.8␣-1.35␣-1.8"
             LowerRight="1.8␣-1.35␣-1.8"
             UpperRight="1.8␣␣1.35␣-1.8">
        </Machine>
</Process>
</pvx>
```

Abbildung 3.8: PVX-Datei für LHRD bestehend aus zwei Displays die jeweils von einem Rechner (*Links* und *Vorn*) betrieben werden

## Equalizer

Equalizer [EMP09] ist eine Middleware für die Entwicklung und Ausführung von OpenGL Anwendungen. Mit Equalizer können diese Anwendungen auf unterschiedlich skalierten Systemen, bezüglich Renderleistung oder Displaygröße, ausgeführt werden ohne diese zu verändern. Die Anpassung an die verschiedenen System erfolgt über die Konfiguration.
Die Konfiguration für Equalizer Anwendungen liegt als Text in EQC-Dateien vor. In diesen sind die Einstellungen für die Server gespeichert. Dabei werden in einem Teil für jeden Rechner Ausgabefenster mit Pixelkoordinaten definiert, denen ein Channel zugewiesen wird (Abb. 3.9 oben). Im Canvas-Teil der Konfiguration werden dann die physischen Positionen der Displays im Raum definiert, auf die in den Segmenten mit Verweis auf die Channels die Ausgabefenster abgebildet werden (Abb. 3.9 unten).
Equalizer habe ich auf keiner der Hardware Installationen getestet sondern habe mich hier lediglich auf die Konfigurationsdatei konzentriert. Dies lag daran, dass die Entscheidung Equalizer mit in die Liste der zu unterstützender Software erst nach der Anforderungsanalyse gefallen ist.

```
window{ name "left"
                viewport [ 100 100 480 300 ]
                channel{name "channel1"}
}
window{ name "right"
                viewport [ 580 100 480 300 ]
                channel{name "channel2"}
}

canvas{ layout 0
                wall{   bottom_left  [ -1.6 -.5 -1 ]
                                bottom_right [  1.6 -.5 -1 ]
                                top_left     [ -1.6  .5 -1 ]
                }
                segment{        channel "channel1"
                                        viewport [ 0 0 .5 1 ]
                }
                segment{        channel "channel2"
                                        viewport [ .5 0 .5 1 ]
                }
}
```

Abbildung 3.9: Ausschnitt aus einer EQC-Textdatei für die Konfiguration von Equalizeranwendungen

# 4 Konzeption

Das Ziel hinter Open Display Enviroment Configuration Language (OpenDECL) ist es, eine adäquate aber dennoch knappe Beschreibung für LHRD zu entwickeln. Diese Beschreibung sollte alle nötigen Informationen enthalten, die für die Konfiguration der Anwendung, die auf dem beschriebenen LHRD ausgeführt werden soll, benötigt werden. Aus der Softwareanalyse ergab sich, dass für eine Konfiguration im wesentlichen zwei Aspekte eine Rolle spielen. Zum einem der Aufbau des LHRD aus einzelnen Displays und der Abbildung der jeweiligen Pixelkoordinaten darauf. Und zum anderem die Infrastruktur die das LHRD betreibt.

## Beschreibung des Displays

Zur Beschreibung des Displays ist zum einem die physische Größe und Anordnung der Display Elemente an sich nötig, als auch die Abbildung der Pixelkoordinaten auf diese Displays.
Unter der Annahme, dass es sich bei den Displays um Rechteckige Flächen handelt, lässt sich die Breite und Höhe einfach angeben. Für die Positionierung der Displays im Raum ist ein Referenzkoordinatensystem notwendig. Wenn man dieses definiert hat, lassen sich alle Displays positionieren, in dem beispielsweise ein Vektor die Position der linken oberen Ecke des Displays beschreibt. Damit hat man das Display positioniert aber noch nicht orientiert. Dafür sind zwei weitere Vektoren notwendig. Mit drei Vektoren, die beispielsweise die drei der vier Ecken des Displays beschreiben, lässt sich ein Display ähnlich wie in den PVX Dateien bei der Konfiguration von ParaView eindeutig im Raum platzieren. Damit lässt sich die Anordnung der einzelnen Displays in einem LHRD beschreiben (Abb. 4.1 gelbes Display).
Die Frage, wo man das Referenzkoordinatensystem ansetzt, also wohin man den Ursprung legt, kann auf zwei verschiedenen Wegen geklärt werden. Wenn man kein anderes Koordinatensystem als Anhaltspunkt hat, zum Beispiel durch ein Trackingsystem oder ähnliches, kann man den Benutzer als Ursprung des Referenzkoordinatensystems verwenden. Dazu sucht man sich die bevorzugte Position des Benutzers vor dem LHRD und beschreibt dann ausgehend vom Kopf des Benutzers als Koordinatenursprung die Positionierung und Orientierung der einzelnen Displays (Abb. 4.1 oben).
Im Fall eines bereits vorhandenen Koordinatensystems, beispielsweise durch ein Trackingsystem, können die einzelnen Displays in diesem System positioniert und orientiert werden. In diesem Fall wäre jedoch angebracht auch den Benutzer zu positionieren, da sich dessen Position nicht mehr durch den Koordinatenursprung, wie in dem ersten Fall, ergibt. Der Benutzer kann mit zwei Vektoren beschrieben werden, die die Position und Orientierung seines Kopfes beschreiben (Abb. 4.1 unten).
Besonders bei LHRD, die einen erweiterten Desktop abbilden, ist es wichtig auch die Zuordnung der Pixelkoordinaten zu den einzelnen Displays zu beschreiben. Diese Beschreibung ist nötig, da die Ausgabefenster der Anwendungen für das LHRD nur in diesem Pixelraum positioniert werden können. Die naheliegende Lösung dafür sind drei weitere Vektoren zur Beschreibung der einzelnen Displays. Diese Vektoren sind keine Raumvektoren, sondern Pixelvektoren und beschreiben die entsprechenden Ecken des Displays in Pixelkoordinaten (Abb. 4.1 blaues Display).
Mit diesen beiden Beschreibungen der Ecken der einzelnen Displays, also räumliche Koordinaten und Pixelkoordinaten, lässt sich das LHRD eindeutig beschreiben und die Anwendungen dafür konfigurieren.

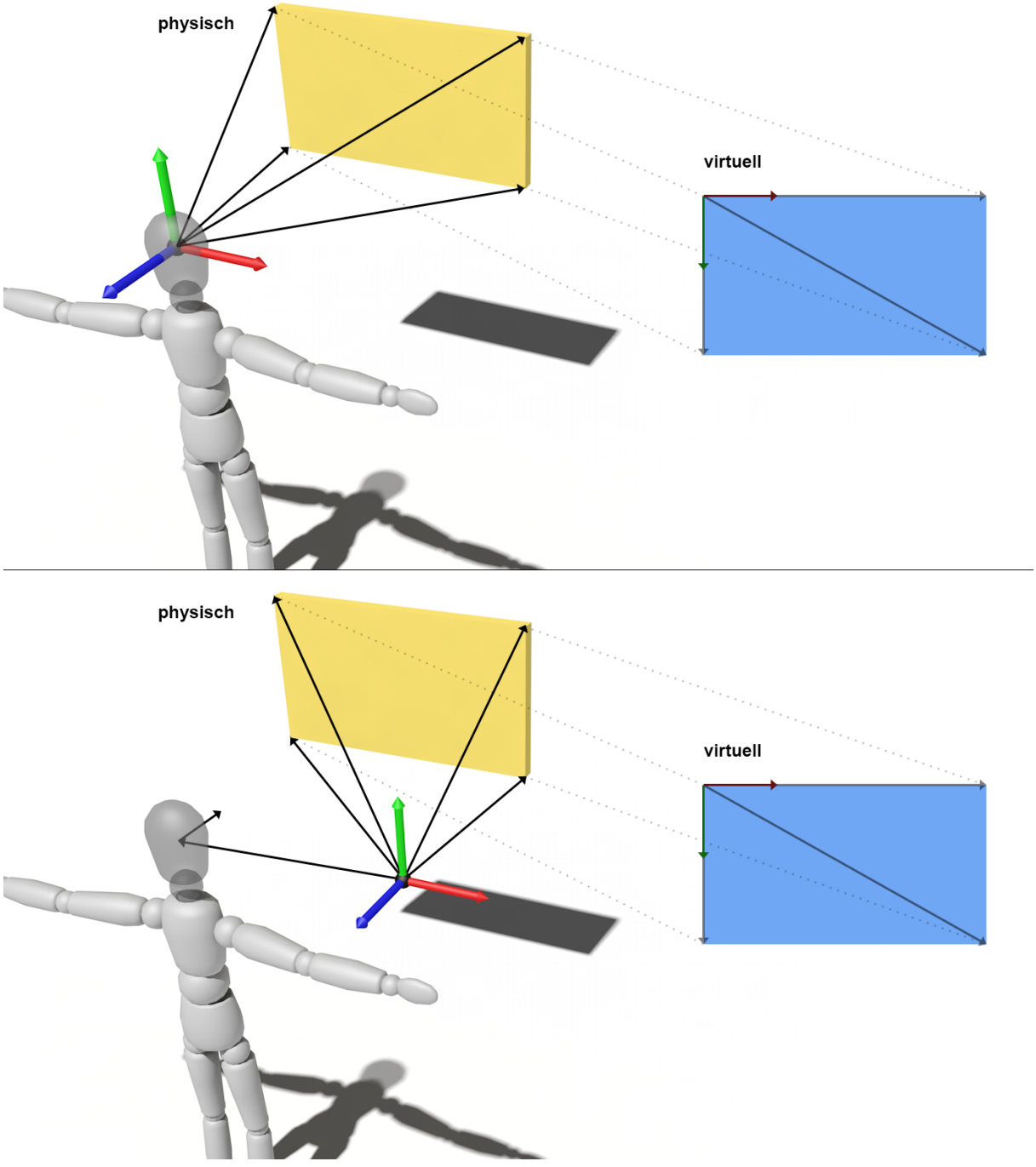


Abbildung 4.1: Konzept zur Beschreibung der physischen (gelb) und virtuellen (blau) Position eines Displays
oben: Ursprung des physischen Koordinatensystems liegt im Nutzer
unten: Ursprung des physischen Koordinatensystems ist getrennt vom Nutzer

## Beschreibung der Infrastruktur

Die Beschreibung der Infrastruktur ist besonders dann wichtig, wenn mehrere Rechner das LHRD betreiben. Dann ist es wichtig zu wissen, welcher Teil des LHRD von welchem Rechner betrieben wird. Dies lässt sich beschreiben, indem man für jeden Rechner angibt, mit welchen Displays er verbunden ist. Zusammen mit der Beschreibung der Displays, lässt sich so einfach definieren, welcher Rechner für welchen Teil des LHRD zuständig ist.
Bezüglich der Verbindung der Rechner untereinander, gehe ich von einem Ethernet Netzwerk aus, was von allen Rechnern geteilt wird. Da es in solchen Netzwerken keine gerichteten Verbindungen gibt, genügt es zu beschreiben, in welchen Netzwerken sich die Rechner befinden und welche Adressen sie darin haben.

# 5 OpenDECL-Spezifikation

Die aus meiner Konzeption hervorgehende benötigten Beschreibungen für LHRD, habe ich in die Spezifikation der OpenDECL Dokumente umgesetzt. Diese Spezifikation liegt als XML Schema Dokument auf dem beigelegtem Datenträger vor und besteht im groben aus zwei Teilen. Zum einem die Beschreibung der Infrastruktur mit den *node* und *network* Elementen und der Beschreibung des LHRD in den *display-setup* Elementen. Ein gültiges OpenDECL Dokument benötigt dabei beliebig viele aber mindestens ein *node* Element, beliebig viele *network* Elemente und beliebig viele aber mindestens ein *display-setup* Element. Abbildung 5.1 zeigt einen Überblick, wie die einzelnen Elemente verschachtelt sind und auf welche Elemente die Referenzen verweisen. Im folgenden werden die einzelnen Elemente und deren Attribute der Spezifikation näher beschrieben und welche Aspekte der Konzeption damit umgesetzt werden.

## 5.1 node

Das *node* Element beschreibt einen Rechner. Würde das LHRD also von einem Cluster betrieben, gäbe es in der OpenDECL Beschreibung dafür auch mehrere *node* Elemente. Das *id* Attribut beschreibt einen eindeutigen Bezeichner für den Rechner und ist ein Pflichtattribut. Dies kann beispielsweise der Name des Rechners im Netzwerk sein. Das *purpose* Attribut ist optional und kann die Funktion des Rechners beschreiben. Solche Funktionen können zum Beispiel “Rendern“, “Verarbeitung“ oder “Berechnung“ sein.
Ein *node* Element und damit ein Rechner, kann beliebig viele *graphics-device* und *network-device* Elemente enthalten.

### 5.1.1 graphics-device

Das *graphics-device* Element beschreibt eine Grafikkarte in einem Rechner. Das *id* Attribut dient als eindeutiger Bezeichner für diese Grafikkarte und ist ein Pflichtattribut. Das *gpu-count*, *vram* und *model-name* Attribut sind optionale Attribute um die Grafikkarte bezüglich Anzahl der GPUs, Speicher und Modellnamen näher zu beschreiben.
Ein *graphics-device* Element enthält beliebig viele aber mindestens ein *port* Element. Dies schließt zwar nVidia Tessla-Karten aus, welche nur GPU-Computing leisten und keinen Displayanschluss haben. Da sich aber diese Spezifikation auf die Displaybeschreibung konzentriert, habe ich diesen Fall außer Acht gelassen.

#### port

Das *port* Element beschreibt einen Anschluss an der Grafikkarte, an das ein Display angeschlossen werden kann. Das *id* Attribut dient als eindeutiger Bezeichner dieses Anschlusses und dient als Referenz ID für die Zuordnung von Displays an diesen Anschluss. Das *type* und *slot* sind optionale Attribute um diesen Anschluss näher zu beschreiben, beispielsweise ob es sich um einen HDMI oder VGA Anschluss handelt oder welcher Anschluss bei Grafikkarten mit mehreren Anschlüssen beschrieben wird.

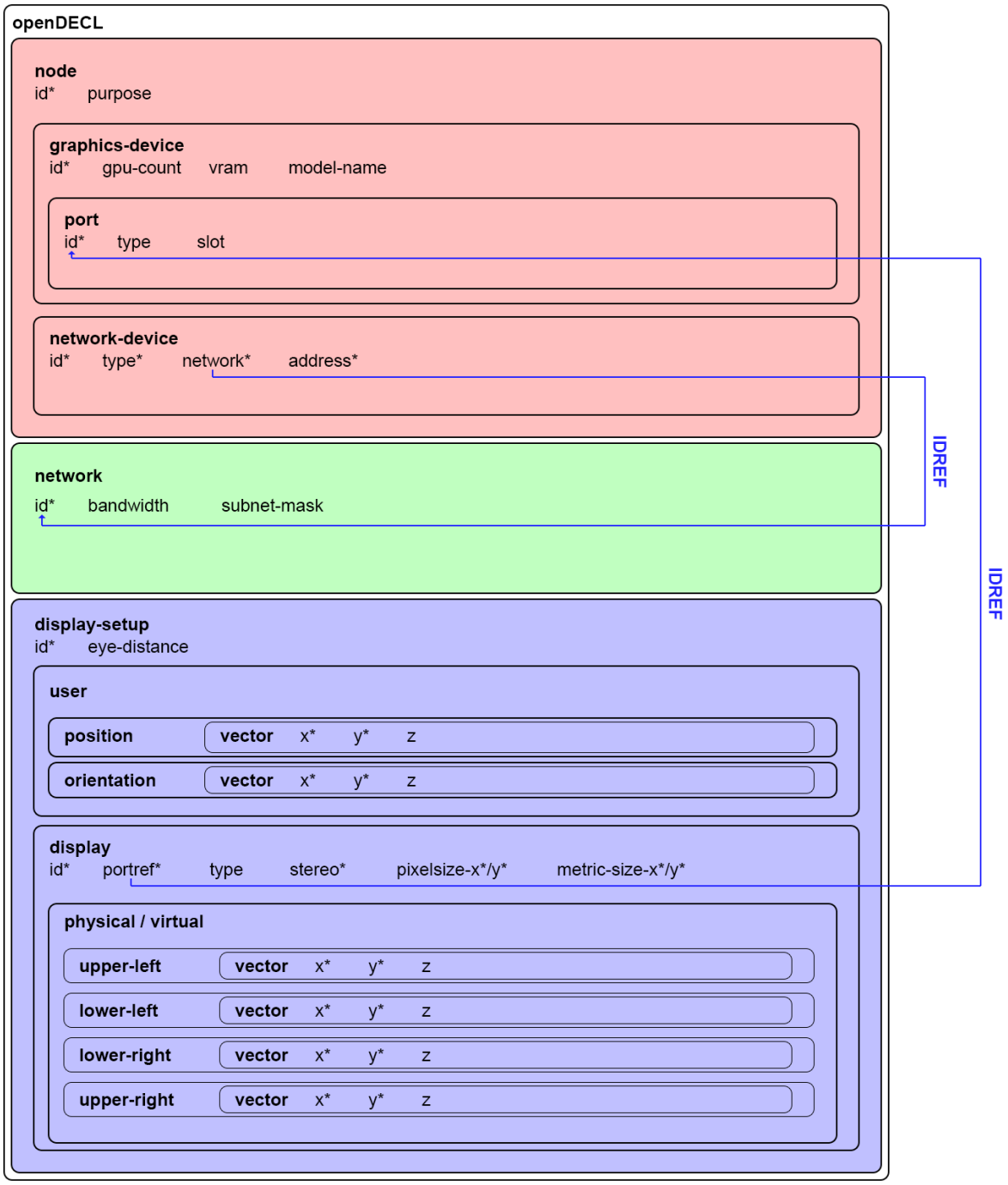


Abbildung 5.1: Vereinfachtes Schema der XML-Spezifikation von OpenDECL mit eingezeichneten ID-Referenzen

### 5.1.2 network-device

Das *network-device* Element beschreibt einen Netzwerk Adapter, der mit einem Netzwerk verbunden ist. Das *id* Attribut dient als eindeutiger Bezeichner dieses Netzwek Adapters und ist, wie alle anderen Attribute von *network-device*, ein Pflichtattribut. Das Attribut *network* ist eine Referenz auf ein *network* Element mit der angegebenen *id*. Damit ist dieser Netzwerk Adapter und damit auch der Rechner in dem dieser angegeben ist, mit dem angegebenen Netzwerk verbunden. Das Attribut *address* gibt dabei die entsprechende Adresse des Netzwerk Adapters im Netzwerk an.

## 5.2 network

Das *network* Element beschreibt ein Netzwerk für mehrere Rechner. Das *id* Attribut dient als eindeutiger Bezeichner des Netzwerks und ist ein Pflichtattribut. Es dient als Referenz ID für die Zuordnung von Netzwerk Adapter die mit diesem Netzwerk verbunden sind. Die Attribute *bandwidth* und *subnet-mask* sind optionale Attribute und dienen zur näheren Beschreibung des Netzwerkes.

Ein Beispiel für die Beschreibung der Infrastruktur mittels *node* und *network* sieht man in Abb. 5.2. Dies ist die Beschreibung des Desktop System.

```
<node id="Ronald-PC" purpose="render">
        <graphics-device id="video1" gpu-count="1" vram="512"
        model-name="ATi␣Radeon␣HD␣3650␣mobility">
                <port type="display" id="M1" slot="onboard"></port>
                <port type="display" id="M2" slot="HDMI␣1"></port>
        </graphics-device>
        <network-device type="ethernet" id="EA1" network="e1" address="127.0.0.1" />
</node>
<network id="e1" bandwidth="10␣Gbit/s" subnet-mask="255.255.255.0" />
```

Abbildung 5.2: Beschreibung der Infrastruktur des Desktop System

## 5.3 display-setup

Das Element *display-setup* stellt eine mögliche Konfiguration von Displays dar. Sind die Displays auf verschiedenen Arten konfigurierbar, so lässt sich dies mit mehreren *display-setup* Elementen abbilden. Das Attribute *id* dient als eindeutiger Bezeichner und ist ein Pflichtattribut. Das Attribut *eye-distance* beschreibt den Augenabstand der für die Stereoprojektion verwendet wird. Wird dieser nicht angegeben, so wird standardmäßig der Wert *0.0* angenommen.
Ein *display-setup* Element kann ein *user* Element enthalten, sowie mindestens ein oder beliebig viele *display* Elemente.

### vector

Das Element *vector* stellt einen zwei- oder dreidimensionalen Vektor dar. Dieser wird für die Beschreibung des Benutzer im Element *user*, sowie für die physische und virtuelle Positionierung der Displays im Element *display* verwendet. Mit den Attributen $x$, $y$ und $z$ lassen sich die Komponenten des Vektors angeben. Dabei sind $x$ und $y$ Pflichtattribute und $z$ optional. Damit lässt sich das *vector* Element zur Darstellung für zwei- und dreidimensionale Vektoren verwenden.

### 5.3.1 user

Das Element *user* stellt den Benutzer in der aktuellen Konfiguration dar. Wird der Benutzer mit diesem Element nicht beschrieben, wird angenommen, dass er sich im Ursprung des angenommenen Koordinatensystem befindet.
Das Element *user* muss ein *position* und ein *orientation* Element enthalten, mit denen der Benutzer beschrieben wird. Diese beiden Elemente müssen jeweils ein *vector* Element enthalten. Dabei beschreibt das Element *position* die Position des Benutzers und das Element *orientation* die Blickrichtung des Benutzers an dieser Position im angenommenen Koordinatensystem.

### 5.3.2 display

Das Element *display* stellt eine Displayfläche dar. Dabei wird nur die reine darstellende Fläche beschrieben. Das heißt im Falle eines Projektors, die Fläche die von diesem bestrahlt wird und im Falle eines Monitors ist dies der Bildschirm ohne den Rahmen. Das Attribut *id* dient als eindeutiger Bezeichner und ist ein Pflichtfeld. Das heißt auch, dass das selbe Display in einem anderem Display-Setup eine andere ID bekommen muss. Mit dem Pflichtattribute *portref* referenziert man auf das *id* Attribut eines *port* Elementes. Damit stellt man die Verbindung dieses Displays mit einem Port und damit einem Rechner dar. Mit den Pflichtattributen *pixel-size-x*,*pixel-size-y*,*metric-size-x* und *metric-size-y* wird die Auflösung und physische Größe des Displays beschrieben. Wenn es sich um ein Display für eine stereoskopische Abbildung handelt, kann mit dem Pflichtattribute *stereo* angegeben werden, für welches Auge das Display abbildet (*left-eye* order *right-eye*). Wenn es sich nicht um eine stereoskopische Abbildung handelt, wird dies mit mit *none* angegeben. Mit dem optionalen Attribut *type* kann das Display näher beschrieben werden, zum Beispiel ob es sich um einen Monitor oder Projektor handelt.
Das *display* Element muss jeweils ein *physical* und ein *virtual* Element enthalten.

#### physical und virtual

Mit dem *physical* Element wird die Position des Display im angenommenen Koordinatensystem beschrieben. Jedes *physical* Element muss jeweils ein *upper-left*,*lower-left*,*lower-right* und *upper-right* Element enthalten, welche wiederum jeweils ein *vector* Element enthält. Diese Elemente beschreiben jeweils die vier Ecken des Displays im Raum. Demnach muss es sich bei den verwendeten *vector* Elementen um dreidimensionale Vektoren handeln.
Mit dem *virtual* Element wird die Abbildung der Pixelkoordinaten auf das Display beschrieben. Dies geschieht analog zum *physical* Element, in dem jeweils die vier Ecken des Displays mit ihren Pixelkoordinaten beschrieben werden. Der Unterschied besteht darin, das die verwendeten *vector* Elemente zweidimensionale Vektoren beschreiben.
Auch wenn nur drei *vector* Elemente nötig wären, um das Display Physisch und Virtuell zu beschreiben, werden alle vier Ecken angegeben. Damit erübrigt sich das Errechnen der nicht angegebenen Ecke, sondern man kann diese direkt aus dem Dokument auslesen. Der Nachteil besteht darin, dass somit auch vier Ecken angegeben werden können, die nicht in einer Ebene liegen. Dies muss bei der Erstellung, zum Beispiel im Editor, beachtet werden.

Ein Beispiel für die Beschreibung einer Konfiguration mit Benutzer und einem von zwei Displays sieht man in Abb. 5.3. Die kompletten Beschreibungen der Installationen mittels openDECL Dokumenten befinden sich auf dem beigelegten Datenträger. Dabei habe ich bei den Größen- beziehungsweise Auflösungsangaben in den Beschreibungen und Vektoren jeweils Meter und Pixel verwendet.

```
<display-setup id="flat">
        <user>
                <position>
                        <vector x="0" y="0" z="0"></vector>
                </position>
                <orientation>
                        <vector x="0" y="0" z="-1"></vector>
                </orientation>
        </user>
        <display id="Mon1" portref="M1" type="monitor" stereo="none"
        pixel-size-x="1280" pixel-size-y="800" metric-size-x="0.32" metric-size-y="0.2">
                <physical>
                        <upper-left>
                                <vector x="-0.32" y="0.1" z="-0.5"></vector>
                        </upper-left>
                        <lower-left>
                                <vector x="-0.32" y="-0.1" z="-0.5"></vector>
                        </lower-left>
                        <lower-right>
                                <vector x="0" y="-0.1" z="-0.5"></vector>
                        </lower-right>
                        <upper-right>
                                <vector x="0" y="0.1" z="-0.5"></vector>
                        </upper-right>
                </physical>
                <virtual>
                        <upper-left>
                                <vector x="0" y="0"></vector>
                        </upper-left>
                        <lower-left>
                                <vector x="0" y="800"></vector>
                        </lower-left>
                        <lower-right>
                                <vector x="1280" y="800"></vector>
                        </lower-right>
                        <upper-right>
                                <vector x="1280" y="0"></vector>
                        </upper-right>
                </virtual>
        </display>
```

Abbildung 5.3: Beschreibung einer Konfiguration mit Benutzer und einem Display

# 6 XSLT-Konfigurationsgenerierung

Um aus den OpenDECL-Beschreibungen der Installationen die Konfigurationen für die entsprechenden Anwendungen zu generieren, wird XSLT in Verbindung mit XPath verwendet. Mit XSLT-Dokumenten lassen sich XML-Dokumente in andere XML-Dokumente transformieren. XPath wird dabei genutzt um im Dokumentenbaum des Ausgangsdokumentes Elemente zu adressieren. Dabei gibt man die Elemente des Zieldokument an und fügt darin die benötigten Informationen aus der OpenDECl-Beschreibung ein (siehe Beispiele für ParaView Konfiguration). Für Konfigurationen die kein XML-Dokument sind, kann man mittels XSLT auch Textteile mit den Informationen aus der OpenDECL-Beschreibung verküpfen (siehe Beispiel für MegaMol Konfiguration an der Stereowall).Im Ergebnisdokument muss dann lediglich das Prolog-Element, welches das Dokument als XML-Dokument kennzeichnet, entfernt werden (Abb. 6.1 erste Zeile). Wenn in den OpenDECL-Beschreibungen mit Namespaces, insbesondere dem von OpenDECL selbst, gearbeitet wird, muss dies im XSLT-Dokument berücksichtigt werden. Dazu muss der gleiche Namespace im XSLT-Dokument mit einem Prefix definiert und dann in den entsprechenden XPath Ausdrücken verwendet werden (siehe Abb. 6.1 dritte Zeile und Beispiele). Wenn dabei im Ergebnisdokument ebenfalls XML-Elemente verwendet werden, muss mit einem weiteren Befehl verhindert werden, dass diese Elemente ebenfalls den Prefix des zuvor geladenen Namespace haben (Abb. 6.1 vierte Zeile).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:openDECL="http://www.tu-dresden.de"
        exclude-result-prefixes="openDECL">
```

Abbildung 6.1: Kopf der XSLT-Dokumente zur Generierung der Konfigurationen aus OpenDECL-Beschreibungen

Die Komplexität der XSLT-Dokumente hängt stark von den Möglichkeiten der XPath-Ausdrücke ab, da diese Ausdrücke bestimmen, welche Elemente der OpenDECL-Beschreibung in das Zieldokument übernommen werden. Wichtig ist dabei zu wissen, wie viel von der OpenDECL-Beschreibung bekannt ist und sich im XSLT-Dokument wieder findet. Will man beispielsweise mit ein und dem selben XSLT-Dokument mehrere ähnliche OpenDECL-Dokumente verarbeiten, sollten im XSLT-Dokument keine Bezeichner (zum Beispiel IDs) verwendet werden, die nicht in allen OpenDECL-Dokumenten verwendet werden. XSLT bietet die Möglichkeit bedingter Anwendung von Regeln (xsl:choose, xsl:if). Damit lassen sich sehr komplexe XSLT-Dokumente erstellen, die auch für mehrere OpenDECL-Dokumente verwendet werden können um Konfigurationen für eine Anwendung zu erstellen. Neben den Befehlen zum Adressieren der Elemente im Dokumentenbaum, bietet XPath auch eine Reihe von erweiterten Funktionen, wie zum Beispiel das Aufsummieren mehrerer Werte oder das Bestimmen von minimalen und maximalen Werten einer Reihe von Werten. Damit lassen sich komplexe Ausdrücke formen um beispielsweise die Anzahl der Displays zu bestimmen oder die maximale physische Ausdehnung des Displaysetups. Bei komplexeren Berechnungen stößt XPath jedoch an seine Grenzen. Solche komplexeren Berechnungen kann Beispielsweise der Test sein, ob alle Displays in einer Ebene liegen, oder die Berechnung eines Winkels zwischen zwei Displays

## 6.1 Beispiele

In den hier vorgestellten Beispielen habe ich mich wegen der besseren Übersichtlichkeit darauf beschränkt, dass die XSLT-Dokumente lediglich für eine OpenDECL-Beschreibung bestimmt sind. Dies sind die entsprechenden OpenDECL-Beschreibungen der von mir analysierten Installationen, die sich auch auf dem beigelegtem Datenträger befinden, ebenso wie die hier vorgestellten XSLT-Dokumente. In allen Beispielen wird die Konfiguration ausgehend von einem Displaysetup, dessen ID bekannt ist, erstellt.

### MegaMol-Konfiguration für Stereowall

Bei diesem Beispiel handelt es sich um ein XSLT-Dokument, welches anhand der OpenDECL-Beschreibung der Stereopowerwall der Professur für Computergraphik und Visualisierung die Konfiguration für MegaMol generiert. Als Ergebnisdokument wird ein Text generiert, der die beiden XML-Elemente zum bestimmen der Fenster für *megamol.cfg* enthält, sowie darunter die Informationen bezüglich der tileview für die Parameterdatei. Diese beiden Teile können kopiert und an den entsprechenden Stellen eingefügt werden.
Dies ist ein recht einfaches Beispiel, bei dem bekannt ist, das beide Displays auf ein und der selben Fläche liegen. Mit den IDs der Displays wird auf deren *pixel-size* Attribute zugegriffen sowie die virtuellen Koordinaten der Ecken (Abb. 6.2 oben). Bei der Stereo Konfiguration für die TileView-Module wird in Abhängigkeit des *stereo* Attribut der entsprechende Wert gesetzt (Abb. 6.2 unten).

```
<set name="tile1-window">
<xsl:attribute name="value">
x<xsl:value-of select="openDECL:display[@id='projL']/openDECL:virtual/
openDECL:upper-left/openDECL:vector/@x"/>
y<xsl:value-of select="openDECL:display[@id='projL']/openDECL:virtual/
openDECL:upper-left/openDECL:vector/@y"/>
w<xsl:value-of select="openDECL:display[@id='projL']/openDECL:virtual/
openDECL:lower-right/openDECL:vector/@x"/>
h<xsl:value-of select="openDECL:display[@id='projL']/openDECL:virtual/
openDECL:lower-right/openDECL:vector/@y"/>nd
</xsl:attribute>
</set>


<xsl:choose>
<xsl:when test="openDECL:display[@id='projL']/@stereo = 'left-eye'">
::tile1::TileView1::eye=Left Eye
::tile1::TileView1::projType=Stereo OffAxis
</xsl:when>
<xsl:when test="openDECL:display[@id='projL']/@stereo = 'right-eye'">
::tile1::TileView1::eye=Right Eye
::tile1::TileView1::projType=Stereo OffAxis
</xsl:when>
</xsl:choose>
```

Abbildung 6.2: Teile des XSLT-Dokumentes für die Konfiguration von MegaMol für die Stereowall
oben: das Erstellen des set-Elementes und das setzen der entsprechenden Attribute
unten: das Schreiben der entsprechenden Textteile für die Stereoabbildung abhängig vom *stereo* Attribut

Das komplette XSLT-Dokument befindet sich auf dem beigelegtem Datenträger und kann mit der OpenDECL-Beschreibung der Stereowall benutzt werden (zum Beispiel mit dem Editor).

### ParaView-Konfiguration für MT Powerwall

Dieses Beispiel ist ein XSLT-Dokument, welches mit der OpenDECL-Beschreibung der Powerwall des Lehrstuhl für Multimedia-Technologie, die PVX-Datei für den ParaView-Server erstellt. Dabei werden ausschließlich XML-Elemente generiert, sodass das Ergebnisdokument als PVX-Datei gespeichert und direkt verwendet werden kann.
Dies ist ein etwas komplexeres Beispiel, bei dem angenommen wird, dass der Name des einzigen Rechner, an dem die zwölf Displays angeschlossen sind, bekannt ist. Der Hauptteil, bei dem die Daten aus der OpenDECL-Beschreibung genutzt werden, liegt in der Erstellung der einzelnen *Machine* Elemente für die Ausgabefenster. Dabei wird mit dem *xsl:for-each* Element für jedes *display* Element im ausgewählten Displaysetup, ein *Machine* Element erzeugt (Abb. 6.3). Für dieses werden dann die entsprechenden Attribute gesetzt wobei die entsprechenden Daten aus der OpenDECL-Beschreibung des Displays verwendet werden.
Dieses Beispiel ist unabhängig von den IDs der Displays, da immer alle Displays im Displaysetup ver-

```
<xsl:for-each select="openDECL:display">
<Machine Name="MULTI" Environment="DISPLAY=:0" FullScreen="0" ShowBorders="0">
```

Abbildung 6.3: Teil des XSLT-Dokumentes für die Konfiguration von ParaView für die MT Powerwall Verwendung der for-each Funktion von XSLT

wendet werden. Somit ist es auch Möglich ein weiteres Display-Setup in der OpenDECL-Beschreibung zu definieren, bei der beispielsweise weniger Displays verwendet werden. Das XSLT-Dokument könnte dann auch erfolgreich mit dieser Beschreibung verwendet werden, es muss lediglich die ID des Displaysetups angepasst werden.
Das komplette XSLT-Dokument befindet sich auf dem beigelegtem Datenträger und kann mit der OpenDECL-Beschreibung der MT-Powerwall benutzt werden (zum Beispiel mit dem Editor).

### ParaView-Konfiguration für CAVE

In diesem Beispiel handelt es sich um ein XSLT-Dokument, welches anhand der OpenDECL-Beschreibung der CAVE des Lehrstuhl Konstruktionstechnik/CAD die PVX-Dateien für die ParaView Server generiert.
Dieses Beispiel ist ähnlich zu dem vorherigen. Auch hier wird für jedes *display* Element ein entsprechendes *Machine* Element angelegt und die entsprechenden Attribute gesetzt. Der Unterschied zur MT-Powerwall besteht darin, dass die Displays nicht an einem Rechner hängen, sodass das *Name* Attribut den Namen des entsprechenden Rechners haben muss, der das Display betreibt. Dies geschieht durch einen komplexen XPath Ausdruck, der noch einmal die Mächtigkeit von XPath für diese Zwecke unterstreicht (Abb. 6.4). Zunächst werden die *graphics-device* Elemente von allen *node* Elementen genommen (Abb. 6.4 dritte Zeile). Dann wird das *graphics-device* Element gewählt, dessen *id* Attribut mit dem *portref* Attribut des aktuellen *display* Elementes übereinstimmt (Abb. 6.4 vierte Zeile). Nun wird das *id* Attribut des *node* Elements genommen, in dem sich das entsprechende *graphics-device* Element befindet, und als Wert für das *Name* Attribute verwendet (Abb. 6.4 fünfte Zeile). Dabei wird angenommen, dass die Namen der Rechner den *id* Attributen ihres entsprechenden *node* Elementes in der OpenDECL-Beschreibung entsprechen.
Das komplette XSLT-Dokument befindet sich auf dem beigelegtem Datenträger und kann mit der OpenDECL-Beschreibung der CAVE benutzt werden (zum Beispiel mit dem Editor).

```
<xsl:attribute name="Name">
<xsl:value-of
select="../../openDECL:node/openDECL:graphics-device
/openDECL:port[@id=current()/@portref]
/../../@id" />
</xsl:attribute>
```

Abbildung 6.4: Teil des XSLT-Dokumentes für die Konfiguration von ParaView für die CAVE
Komplexer XPath-Ausdruck in select von xsl:value of, welcher den Namen des Rechners ermittelt, der das aktuell zu verarbeitende Display betreibt

# 7 Editor

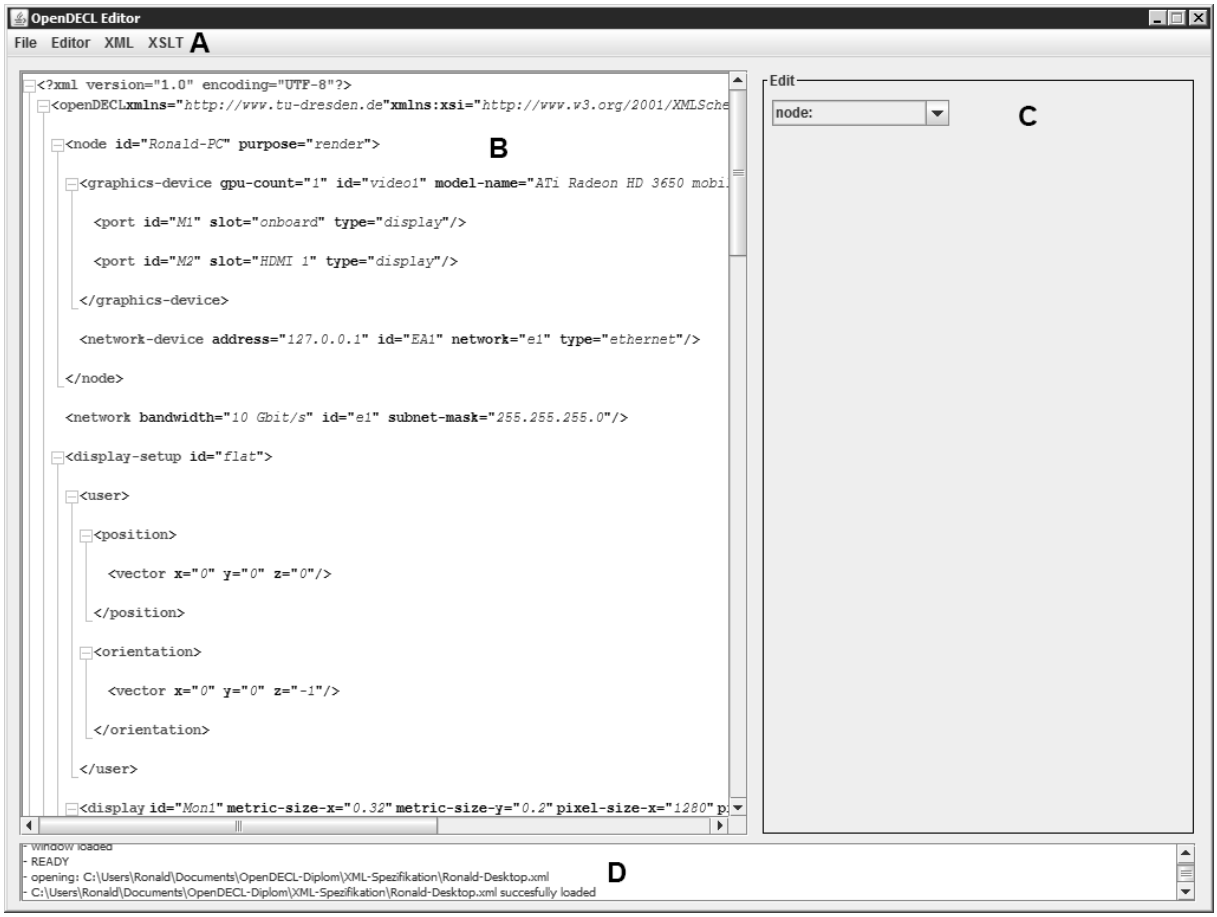


Abbildung 7.1: Editor zum Bearbeiten von OpenDECL-Dokumenten A, Menüleiste; B, Ansicht des aktuellen Dokumentes; C, Interface zum Bearbeiten des Dokumentes; D, Statuszeile

Bei dem Editor handelt es sich um einen Prototypen zum Editieren von OpenDECL-Dokumenten (Abb. 7.1). Es werden Funktionen zum editieren aller in der Spezifikation vorgestellten Elemente und Attribute zur Verfügung gestellt. Außerdem kann das Dokument auf Wohlgeformtheit getestet und anhand einer XML-Schema Datei validiert werden. Für den Export von Konfigurationen steht die Funktion zum anwenden einer XSLT-Datei zur Verfügung. Während der Bearbeitung kann das Dokument in einen Status kommen, in der es nicht mehr valide bezüglich des OpenDECL-Schema ist. Daher sollte vor dem Speichern immer eine Validierung durchgeführt werden.

## 7.1 Funktionen

Um ein OpenDECL-Dokument zu bearbeiten, muss zunächst ein gespeichertes Dokument geladen werden oder ein neues angelegt werden, wobei dabei ein Standarddokument mit minimalem Inhalt geladen wird. Beides kann über File in der Menüleiste gemacht werden (Abb. 7.1 A). Das Dokument erscheint in der Ansicht links (Abb. 7.1 B) wobei die XML-Syntax hervorgehoben wird und die einzelnen Elemente

verkleinert oder expandiert werden können.
Zum Bearbeiten wählt man rechts eines der drei Hauptelemente (*node*, *network* oder *display-setup*) aus dem Dropdown-Menü (Abb. 7.1 C) aus. Dadurch verändert sich entsprechend der Auswahl das Interface zum Bearbeiten. Es erscheinen Optionen zum Bearbeiten der entsprechenden Unterelemente und Attribute. Für Elemente gibt es Dropdown-Menüs, in denen diese mit ihren IDs aufgeführt. Zusätzlich gibt es mit *add new* immer die Option ein neues Element anzulegen beziehungsweise mit dem x hinter dem Dropdown-Menü das ausgewählte Element zu löschen. Unter den Dropdown-Menüs befinden sich Textfelder zum Editieren der Attribute. Bei bereits vorhandenen Elementen werden diese mit den Werten der Attribute vorbelegt. Pflichtattribute sind mit einem *-Zeichen gekennzeichnet. Unter den Textfeldern für Attribute gibt es einen Set-Button. Mit einem klick auf diesen Set-Button, werden die editierten Attribute des Elementes im aktuellen Dokument verändert beziehungsweise ein neues Element mit den angegebenen Attributen im aktuellen Dokument angelegt. Dabei wird kontrolliert ob alle Pflichtattribute gesetzt sind und im Falle von IDs ob diese eindeutig sind.
Möchte man den Text des aktuellen Dokumentes direkt bearbeiten, kann über Editor in der Menüleiste die Dokumentenansicht umgeschaltet werden. Dabei kann zwischen XML-Highlight-Ansicht (Standardansicht) und Text-Editor Ansicht umgeschaltet werden. In der Text-Editor-Ansicht lässt sich der Text des Dokumentes direkt bearbeiten und so beispielsweise größere Teile kopieren oder Elemente eines anderen Namespaces einfügen. In der Text-Editor-Ansicht stehen die zuvor beschriebenen Funktionen zum bearbeiten des Dokumentes nicht zur Verfügung. Wechselt man zurück in die XML-Highlight-Ansicht, wird zunächst überprüft, ob der editierte Text einem wohlgeformten XML-Dokument entspricht. Ist dies nicht der Fall, wird ein Fehler mit entsprechenden Informationen ausgegeben, und es wird nicht in die XML-Highlight-Ansicht gewechselt. Sollte man nicht in der Lage sein, das Dokument so zu bearbeiten, dass es Wohlgeformt ist, kann man im Menü File über Load Last Documentstate, die letzte zwischengespeicherte Version des aktuellen Dokumentes laden und kommt zurück in die XML-Highlight-Ansicht.
Über den Menüpunkt XML kann man testen, ob das aktuelle Dokument wohlgeformt ist und ob es valide bezüglich eines XML-Schemas ist. Beim Validieren muss rechts eine XML-Schema Datei geladen werden wogegen das aktuelle Dokument validiert werden soll. Ähnlich verhält es sich mit dem Export von Konfigurationen via XSLT. Diese Funktion erreicht man im Menüpunkt XSLT. Dabei muss man rechts zunächst die XSLT-Datei laden, die für den Export verwendet werden soll und darunter die Zieldatei, in die das Ergebnis der Transformation gespeichert werden soll.
Bei allen Funktionen die das Dokument verändern sowie bei den XML Tests und XSLT Funktionen, wird in der Statuszeile unten (Abb. 7.1 D) ausgegeben, ob die Funktion erfolgreich durchgeführt wurde oder ob es einen Fehler gab. Im Falle eines Fehlers werden dort zusätzlich Informationen zum Fehler angegeben.

## 7.2 Umsetzung

Der Editor wurde in JAVA umgesetzt und liegt als ausführbare JAR-Datei vor. Die Oberfläche wurde dabei mit Elementen des Swing Paketes von JAVA umgesetzt. Der Kern des Editors ist die *OpenDECLdoc* Klasse. Diese repräsentiert eine OpenDECL Dokument und stellt nach außen hin Funktionen zum Bearbeiten zur Verfügung. Innerhalb der Klasse wird das XML-Dokument durch ein DOM-Document-Objekt repräsentiert. Dieses wird beim erstellen einer Instanz von *OpenDECLdoc* erstellt, wobei eine XML-Datei geladen wird, deren Pfad dem Konstruktor mit übergeben wird. Zum Bearbeiten dieses XML-Dokumentes gibt es nichtöffentliche Funktionen, die beispielsweise ein neues Element einfügen oder Attribute bearbeiten. Nach außen hin werden Funktionen zur Verfügung gestellt, die speziell für OpenDECL Elemente und Attribute konzipiert sind, beispielsweise *addGraphicsDevice* zum hinzufügen eines neuen *graphics-device* Elementes oder *setDisplayPortref* um das *portref* Attribut eines *display* Elementes zu setzen (weitere Funktionen Abb. 7.2). Diese öffentlichen Funktionen benutzen die nichtöffentlichen Funktionen um das Dokument entsprechend zu bearbeiten beziehungsweise Informationen des Dokumentes zurückzugeben.

Im Editor wird eine Instanz der *OpenDECLdoc* Klasse gehalten. Die verschiedenen Interface Elemente rufen die entsprechenden öffentlichen Funktionen zum Bearbeiten des Dokumentes auf. Dabei wird nach jeder Bearbeitung eine temporäre Kopie des aktuellen Dokumentes unter dem Namen *temp.output.xml* lokal gespeichert. Die Funktionen zum Validieren beziehungsweise Testen auf Wohlgeformtheit und für den XSLT-Export, sind nicht Teil der *OpenDECLdoc* Klasse, sondern sind in den entsprechenden Interface Elementen umgesetzt.
Der Quellcode für den Editor befindet sich auf dem beigelegten Datenträger.


Abbildung 7.2: Klassendiagramm der *OpenDECLdoc* Klasse mit allen öffentlichen Funktionen

# 8 Diskussion

Mit OpenDECL stellt diese Arbeit ein Hilfsmittel in Form einer Beschreibungssprache vor, mit deren Hilfe man Anwendungen für LHRDs konfigurieren kann. Dabei wurde gezeigt, wie die Beschreibung anhand der LHRD Installation erstellt wird und wie mit Hilfe von XSLT daraus die Konfiguration der Anwendung generiert werden kann.

Beim Anlegen des OpenDECL-Dokumentes wird die Infrastruktur sowie die physische Anordnung der einzelnen Displays im LHRD abgebildet. Die Beschreibung der Infrastruktur ist dabei einfach, da nur die einzelnen Rechner beschrieben werden müssen, sowie die Displays. Die Beschreibung der einzelnen Rechner beschränkt sich dabei auf die für das Betreiben des LHRD wesentlichen Aspekte, die mit *graphics-device* für die Bildausgabe und *network-device* für die Vernetzung abgedeckt werden. Rechner die andere Aufgaben in der Installation haben, können mit dem *purpose* Attribut entsprechend gekennzeichnet werden, aber darüber hinaus, nicht näher beschrieben werden. Die Beschreibung der Displays in den Attributen des *display* Elementes ist ebenfalls auf die wesentlichen Aspekte reduziert, die nötig sind um ein Display zu beschreiben.Mit dem *portref* Attribut findet eine eindeutige Zuordnung von Display zu Port, und damit zu Rechner statt. Bei mehreren Displaysetups ist darauf zu achten, dass das *id* Attribut von einem Display, welches in mehreren Displaysetups verwendet wird, nicht gleich ist. Zwar führt dies dazu, dass es Displays gibt, die mehrere IDs haben, jedoch kann man in solchen Fällen die Konvention einführen, dass Display IDs die ID des Setupdisplays in dem sie sich befinden, als Prefix vor der eigentlichen ID haben. Damit kann man erkennen, dass es sich um das gleiche Display handelt, dass in unterschiedlichen Displaysetups verwendet wird. Jedes Display nur einmal zu Beschreiben und dann im Displaysetup über die ID entsprechend zu referenzieren wäre auch noch eine Möglichkeit. Jedoch erhöht sich mit jeder Referenz die Komplexität der XSLT-Dokumente zum erstellen der Konfigurationen.
Die physikalische Positionierung der Displays im *physical* Element kann eine Herausforderung darstellen. Für LHRDs in denen die Displays in einer Ebene liegen, oder wie im Falle der CAVE senkrecht aufeinander stehen ist es einfach. Man definiert sich ein Koordinatensystem mit Ursprung. Ohne Trackingsystem kann dies der Nutzer sein. Wenn der Nutzer mittig vor dem LHRD steht und darauf schaut, kann man mit dem Abstand zum LHRD und den Größen der einzelnen Displays die Koordinaten der Displayeckpunkte in einem solchen Koordinatensystem einfach bestimmen. Der Ursprung dieses Koordinatensystem wäre der Kopf des Nutzers, wobei bei der Koordinatenbestimmung für die Displayeckpunkte darauf geachtet werden muss, dass es sich um ein Rechtssystem handelt und unter der Annahmen das die Y-Koordinate die Höhe im Raum beschreibt, der Nutzer in negative Z-Richtung schaut um das LHRD zu sehen.
Bei LHRDs, in dem die Displays nicht in einer Ebene liegen, oder rechtwinklig angeordnet sind, ist die Aufnahme der Displayeckpunkte deutlich schwieriger. Wenn man auch hier den Nutzer als Ursprung betrachtet, müsste man die genauen Koordinaten der Displayeckpunkte abmessen beziehungsweise auf Grundlage eines genau bestimmten Displays relativ die anderen Displays berechnen. Bei ungenauen Messungen oder Berechnungen kann es zu Inkonsistenzen kommen, beispielsweise dass die Eckpunkte eines Displays nicht in einer Ebene liegen. In solchen Fällen könnte ein 3D-Model zum Einsatz kommen, in dem die Displays in einem virtuellen dreidimensionalen Raum platziert werden können. Aus diesem 3D-Model können dann die Displayeckpunkte berechnet werden. Ein solches 3D Model wäre eine mögliche Erweiterung des Editors.

Das erstellen der XSLT-Dateien zur Generierung der Konfigurationen für die Anwendungen, kann je nach Anforderung schwierig sein. Bei Konfigurationen die sich im wesentlichen nur auf die Beschrei-

bung der Ausgabe beschränken, kann man problemlos mit XSLT die gesamte Konfiguration ausgeben. Ein Beispiel dafür sind die PVX-Dateien für Paraview, die, wie in Kapitel 6 beschrieben, als fertige Konfiguration verwendet werden. Bei Konfigurationen, die neben der Beschreibung der Displayausgabe noch weitere Aspekte der Anwendung beschreiben, ist es einfacher, nur den für die Displayausgabe relevanten Teil der Konfiguration mit XSLT zu generieren und dann entsprechend in die Gesamtkonfiguration einzusetzen. Ein Beispiel dafür ist die Konfiguration für MegaMol wie in Kapitel 6 beschrieben. Sollte man in diesem Fall dennoch die Gesamtkonfiguration mit generieren wollen, so muss man die für die Displayausgabe irrelevanten Teile der Konfiguration auch über XSLT mit generieren. Sollten sich diese Teile jedoch oft ändern, so muss dies jedes mal im XSLT-Dokument angepasst werden. Ein solches XSLT-Dokument verliert auch an Übersichtlichkeit.
Je nachdem, wie universell ein XSLT-Dokument auf verschiedene OpenDECL-Dokumente angewandt werden soll, spiegelt sich dies in der Komplexität des XSLT-Dokument wieder. Das Beispiel der MegaMol-Konfiguration für die Stereowall aus Kapitel 6 ist einfach aber eingeschränkt auf die konkrete OpenDECL-Beschreibung der Stereowall. Ändert sich eine der IDs für die Displays, zum Beispiel durch ein weiteres Displaysetup, so muss das XSLT-Dokument entsprechend angepasst werden. Universeller ist das XSLT-Dokument für das Beispiel der ParaView-Konfiguration für die Powerwall aus Kapitel 6. Dieses ist unabhängig von den IDs der Displays oder von deren Anzahl. Das gleiche XSLT-Dokument kann für die OpenDECL-Beschreibung einer anderen Powerwall genutzt werden, solange die ID des Setupdisplays die gleiche bleibt, und alle Displays von einem Rechner betrieben werden.
Die nötige Komplexität der XSLT-Dateien hängt vom Anwendungsszenario ab. Arbeitet man mit nur einem Setup, welches sich kaum verändert, kann man mit einem simplen XSLT-Dokument arbeiten, bei dem die Elemente direkt mit ihren IDs benutzt werden. Benötigt man eine möglichst große Austauschbarkeit, so muss man mit XSLT und XPath die Struktur der OpenDECL-Beschreibung stärker berücksichtigen. Man sollte dabei das direkte benutzen von Elementen über ihre ID vermeiden und mit bedingten Anweisungen arbeiten.

Insgesamt eignet sich OpenDECL zur Beschreibung von LHRDs wie anhand der in dieser Arbeit untersuchten Installationen gezeigt wurde. Das erstellen der Konfigurationen aus diesen Beschreibungen wurde erfolgreich für die in dieser Arbeit analysierten Anwendungen getestet.

# 9 Ausblick

Die Spezifikation von OpenDECL wie sie in dieser Arbeit vorgestellt wird, eignet sich für die Beschreibung von LHRDs, die aus einzelnen, flachen Displays bestehen. Die in [SM11], [SM12] und [JGS+07] beschriebenen nicht planaren Displays stellen für OpenDECL noch ein nicht lösbares Problem dar. Eine denkbare Erweiterung für OpenDECL wäre deshalb, die Beschreibung von gewölbten und anderen nicht planaren Displays.
Des weiteren bleibt die mögliche Verzerrung durch Linsen bei Projektor basierenden LHRDs unberücksichtigt im aktuellen Stand von OpenDECL. Dies wäre ein weiterer Aspekt für die Erweiterung der OpenDECL Spezifikation um eine noch bessere Beschreibung zu erhalten.
Für konkrete Trackingdevices besitzt OpenDECL im Moment kein Element. Dafür könnte die Spezifikation um ein Tracking-Device Element erweitert werden, mit dem verschiedene Tracking-Arten beschrieben werden können. Dieses Element müsste auch mit einem *node* Element verknüpft werden, damit beschrieben wird, an welchem Rechner das Trackingsystem angeschlossen ist.

Der OpenDECL-Editor, der Teil dieser Arbeit ist, bietet einfache Funktionen um alle Elemente und Attribute der Spezifikation in einem OpenDECL-Dokument zu bearbeiten. Wie in Kapitel 8 angerissen, wäre eine Erweiterung des Editors vorstellbar, die die physische Anordnung der Display in einem 3d-Model zulässt und daraus die entsprechenden Vektoren berechnet.
Tests auf Konsistenz während der Eingabe können das Erstellen von OpenDECL-Dokumenten komfortabler machen. So könnte bei der Eingabe von ID-Refrenzen, wie beispielsweise bei *portref* oder *network*, getestet werden, ob die entsprechende ID, auf die referenziert wird, vorhanden ist.
Mit dem Editor ist im Moment nur eine Bearbeitung von OpenDECL-Dokumenten möglich, in den OpenDECL als Default-Namespace angegeben ist. Das heißt, dass alle OpenDECL spezifizierten Elemente des Dokumentes ohne Prefix auftreten. Daher ist es naheliegend den Editor dahingehend zu erweitern, dass auch Dokumente bearbeitet werden, in denen OpenDECL als Namespace mit Prefix verwendet wird. Solche Dokumente könnten andere Beschreibungen sein, in denen OpenDECL als Beschreibung der Displaykonfiguration verwendet wird.

# Literaturverzeichnis


[AGL+04] ALLARD, Jérémie ; GOURANTON, Valérie ; LECOINTRE, Loïck ; LIMET, Sébastien ; RAFFIN, Bruno ; ROBERT, Sophie: *FlowVR: A Middleware for Large Scale Virtual Reality Applications*. 2004. – 497–505 S

[AGL05] AHRENS, James ; GEVECI, Berk ; LAW, Charles: 36 ParaView: An End-User Tool for Large-Data Visualization. In: *The Visualization Handbook* (2005), S. 717

[BGA+03] BRESNAHAN, Glenn ; GASSER, Raymond ; ABARAVICHYUS, Augustinas ; BRISSON, Erik ; WALTERMAN, Michael: Building a large-scale high-resolution tiled rear-projected passive stereo display system based on commodity components. In: *Electronic Imaging 2003* International Society for Optics and Photonics, 2003, S. 19–30

[BJH+01] BIERBAUM, Allen ; JUST, Christopher ; HARTLING, Patrick ; MEINERT, Kevin ; BAKER, Albert ; CRUZ-NEIRA, Carolina: VR Juggler: A Virtual Platform for Virtual Reality Application Development. In: *Virtual Reality*, 2001, S. 89–96

[CCF+00] CHEN, Yuqun ; CLARK, Douglas W. ; FINKELSTEIN, Adam ; HOUSEL, Timothy C. ; LI, Kai: Automatic alignment of high-resolution multi-projector display using an un-calibrated camera. In: *Proceedings of the conference on Visualization'00* IEEE Computer Society Press, 2000, S. 125–130

[CCL+01] CHEN, Han ; CLARK, Douglas W. ; LIU, Zhiyan ; WALLACE, Grant ; LI, Kai ; CHEN, Yuqun: Software Environments For Cluster-Based Display Systems. In: *Cluster Computing and the Grid*, 2001, S. 202–211

[CNSD+92] CRUZ-NEIRA, Carolina ; SANDIN, Daniel J. ; DEFANTI, Thomas A. ; KENYON, Robert V. ; HART, John C.: The CAVE: Audio Visual Experience Automatic Virtual Environment. In: *Commun. ACM* 35 (1992), Juni, Nr. 6, S. 64–72. – ISSN 0001–0782

[CNSD93] CRUZ-NEIRA, Carolina ; SANDIN, Daniel J. ; DEFANTI, Thomas A.: Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In: *Annual Conference on Computer Graphics*, 1993, S. 135–142

[EBZ+12] ENDERT, Alex ; BRADEL, Lauren ; ZEITZ, Jessica ; ANDREWS, Christopher ; NORTH, Chris: Designing large high-resolution display workspaces. (2012), S. 58–65

[EMP09] EILEMANN, Stefan ; MAKHINYA, Maxim ; PAJAROLA, Renato: Equalizer: A Scalable Parallel Rendering Framework. In: *IEEE Transactions on Visualization and Computer Graphics* 15 (2009), May/June, Nr. 3, S. 436–452

[FGH02] FIGUEROA, Pablo ; GREEN, Mark ; HOOVER, H. J.: InTml: a description language for VR applications. In: *Web3D / VRML Symposium*, 2002, S. 53–58

[GKM+15] GROTTEL, S. ; KRONE, M. ; MULLER, C. ; REINA, G. ; ERTL, T.: MegaMol – A Prototyping Framework for Particle-based Visualization. In: *Visualization and Computer Graphics, IEEE Transactions on* 21 (2015), Feb, Nr. 2, S. 201–214. – ISSN 1077–2626

[HHN+02] HUMPHREYS, Greg ; HOUSTON, Mike ; NG, Ren ; FRANK, Randall J. ; AHERN, Sean ; KIRCHNER, Peter D. ; KLOSOWSKI, James T.: Chromium: a stream-processing framework for interactive rendering on clusters. In: *ACM Transactions on Graphics* 21 (2002), S. 693–702

[JGS+07] JOHNSON, T. ; GYARFAS, F. ; SKARBEZ, R. ; TOWLES, H. ; FUCHS, H.: A Personal Surround Environment: Projective Display with Correction for Display Surface Geometry and Extreme Lens Distortion. In: *Virtual Reality Conference, 2007. VR '07. IEEE*, 2007, S. 147–154

[JJR+] JEONG, Byungil ; JAGODIC, Ratko ; RENAMBOT, Luc ; SINGH, Rajvikram ; JOHNSON, Andrew ; LEIGH, Jason: Scalable Graphics Architecture for High Resolution Displays. In: *Presented at IEEE Information Visualization Workshop 2005*, S. 10–24

[KBSR07] KÖNIG, Werner A. ; BIEG, Hans-Joachim ; SCHMIDT, Toni ; REITERER, Harald: POSITION-INDEPENDENT INTERACTION FOR LARGE HIGH-RESOLUTION DISPLAYS. (2007)

[KC07] KANG, Yong-Bin ; CHAE, Ki-Joon: XMegaWall: A Super High-Resolution Tiled Display using a PC Cluster. (2007)

[Kre] KREYLOS, Oliver: *Vrui VR Toolkit*. `http://idav.ucdavis.edu/~okreylos/ResDev/Vrui/`. – zuletzt besucht: 14.08.2015

[Mor12] MORELAND, K.: Redirecting research in large-format displays for visualization. In: *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, 2012, S. 91–95

[NSS+06] NI, Tao ; SCHMIDT, Greg S. ; STAADT, Oliver G. ; LIVINGSTON, Mark A. ; BALL, Robert ; MAY, Richard A.: A Survey of Large High-Resolution Display Technologies, Techniques, and Applications. In: *Virtual Reality, IEEE Annual International Symposium*, 2006, S. 223–236

[RFC+03] RODRIGUES, Fábio ; FERRAZ, Rodrigo ; CABRAL, Márcio ; TEUBL, Fernando ; BELLOC, Olavo ; KONDO, Marcia ; ZUFFO, Marcelo ; LOPES, Roseli: Coupling Virtual Reality Open Source Software Using Message Oriented Middleware. (2003)

[SLM04] SCHROEDER, Will J. ; LORENSEN, Bill ; MARTIN, Ken: *The visualization toolkit*. Kitware, 2004

[SM10] SAJADI, Behzad ; MAJUMDER, Aditi: Scalable Multi-view Registration for Multi-Projector Displays on Vertically Extruded Surfaces. In: *Computer Graphics Forum* (2010). – ISSN 1467–8659

[SM11] SAJADI, B. ; MAJUMDER, A.: Autocalibrating Tiled Projectors on Piecewise Smooth Vertically Extruded Surfaces. In: *Visualization and Computer Graphics, IEEE Transactions on* 17 (2011), Sept, Nr. 9, S. 1209–1222. – ISSN 1077–2626

[SM12] SAJADI, B. ; MAJUMDER, A.: Autocalibration of Multiprojector CAVE-Like Immersive Environments. In: *Visualization and Computer Graphics, IEEE Transactions on* 18 (2012), March, Nr. 3, S. 381–393. – ISSN 1077–2626

[WJS04] WANG, Zonghui ; JIANG, Xiaohong ; SHI, Jiaoying: HIVE: a Highly Scalable Framework for DVE. In: *Virtual Reality, IEEE Annual International Symposium*, 2004

[WSJ06] WANG, Zonghui ; SHI, Jiaoying ; JIANG, Xiaohong: *A Scalable HLA-Based Distributed Simulation Framework for VR Application*. 2006. – 361–371 S

[YGH+01] YANG, Ruigang ; GOTZ, David ; HENSLEY, Justin ; TOWLES, Herman ; BROWN, Michael S.: PixelFlex: a reconfigurable multi-projector display system. In: *IEEE Visualization*, 2001, S. 167–174